Scientific Computing with Matlab

Rafael Palacios (Feb/2014)

Contents

1. Introduction to Matlab.
2. Basic data types.
3. Matlab programming.
4. Advanced data types.
5. Code Optimization.
6. Graphical representation.
7. Developing standalone applications.

Contents

Introduction to Matlab

- ¿What is Matlab?
  - Matlab = **Mat**rix **Lab**oratory.
  - Interactive program for mathematical operations and graphical representation
  - Company: The Mathworks Inc (Natick, MA). `http://www.mathworks.com`
  - Created in California by Jack Little and Cleve Moler in 1984, for matrix computation without the need of background in programming (no loops).

Introducción a Matlab

Development environment

+

Programming Language

+

Graphical User Interace

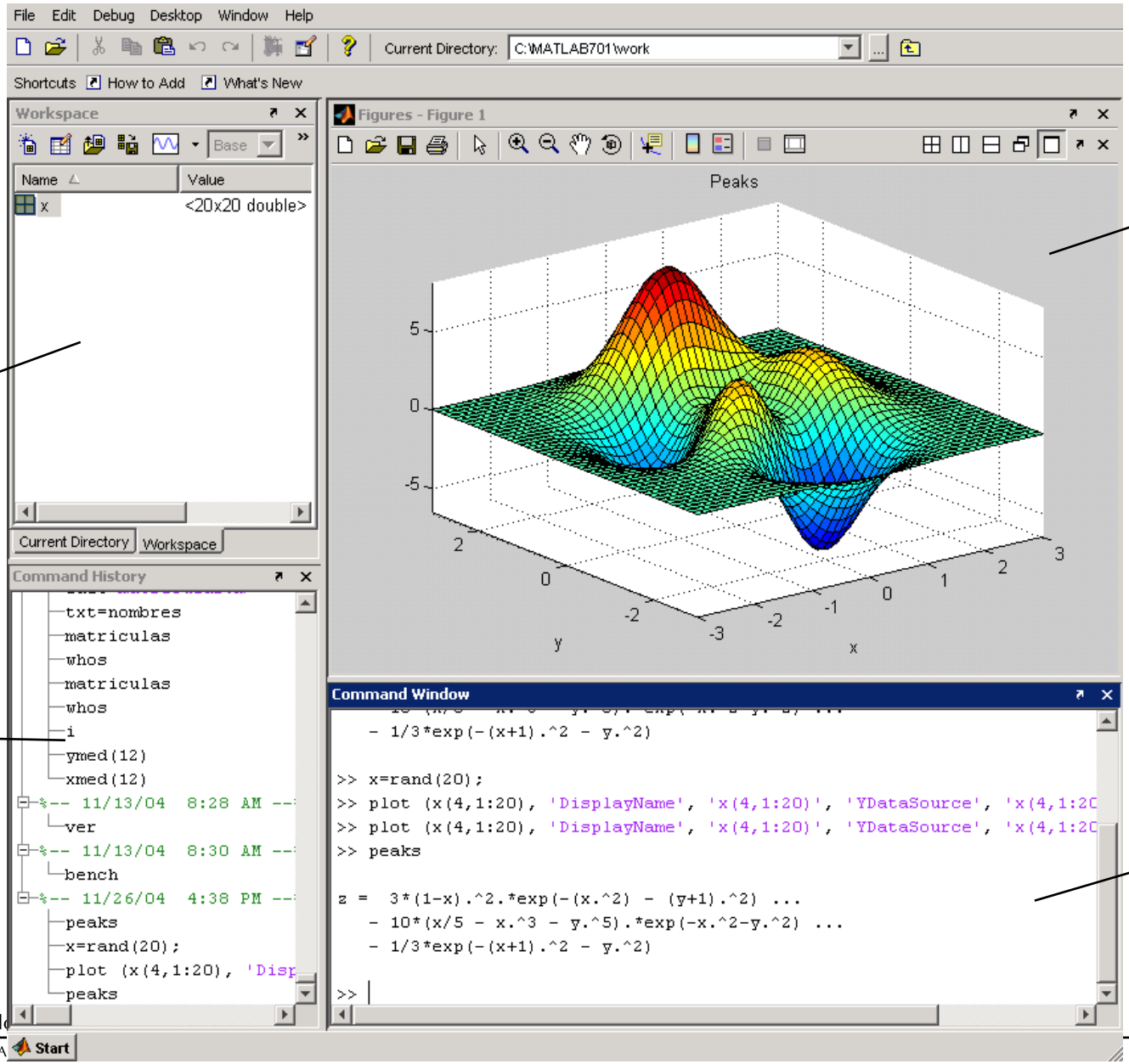
Compatibility

- Platforms that run Matlab
  - Operating systems
    - Unix: Linux, solaris, HP-UX
    - Mac OS X
    - MS-Windows
  - Arquitectures
    - RISC: Sparc, HP-PA
    - PowerMac (G4, G5)
    - Intel Pentium(III, IV, Xeon, M), AMD (Athlon, Opteron)

Matlab versions

- Matlab 5
  - Good graphics (2D, 3D)
  - PC: uses all available memory
- Matlab 6
  - Development environment. Java interface.
  - 3D matrices, structures, cell arrays
- Matlab 7
  - Improvements in environment and Simulink
  - Matlab compiler with OOP
  - Optimized interger computation

Matlab 7 development environment



Workspace

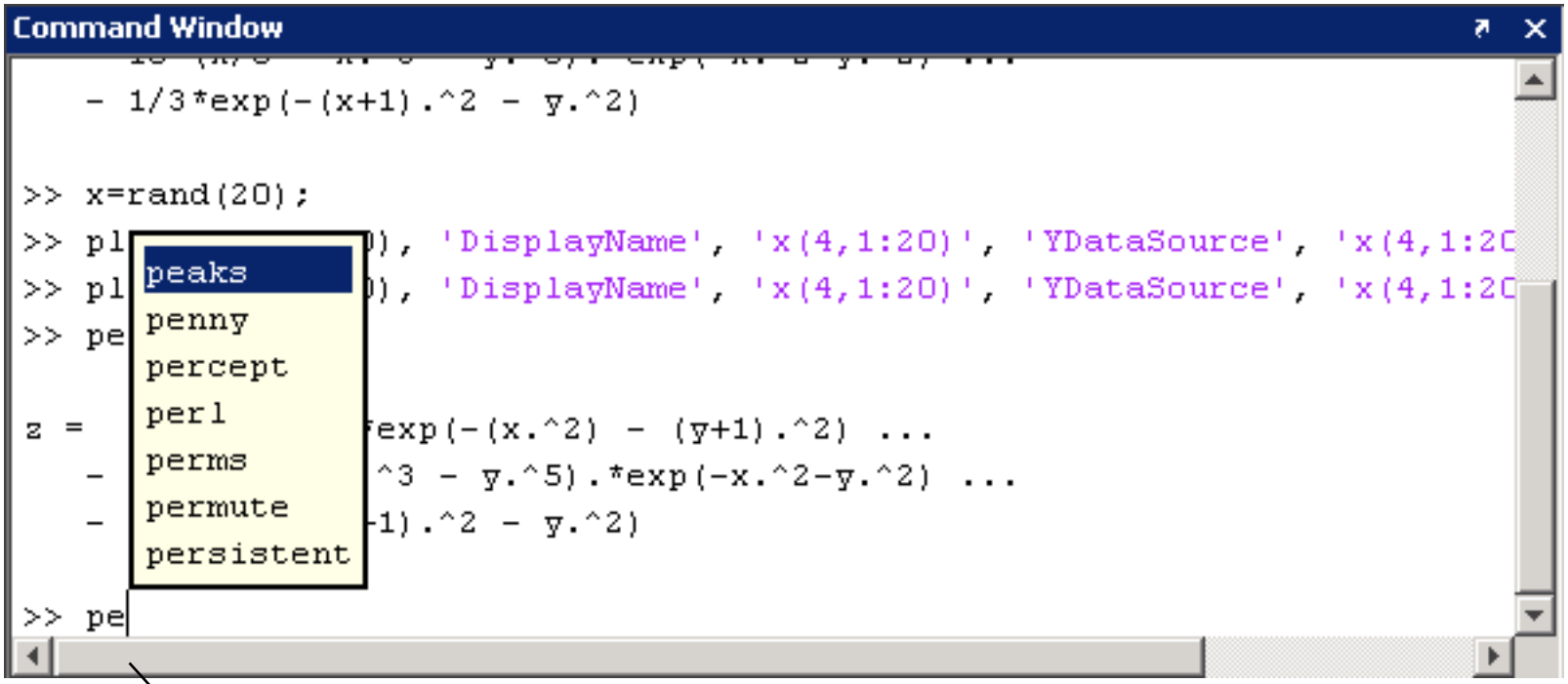
command history

Figuras

command window

Matlab 7

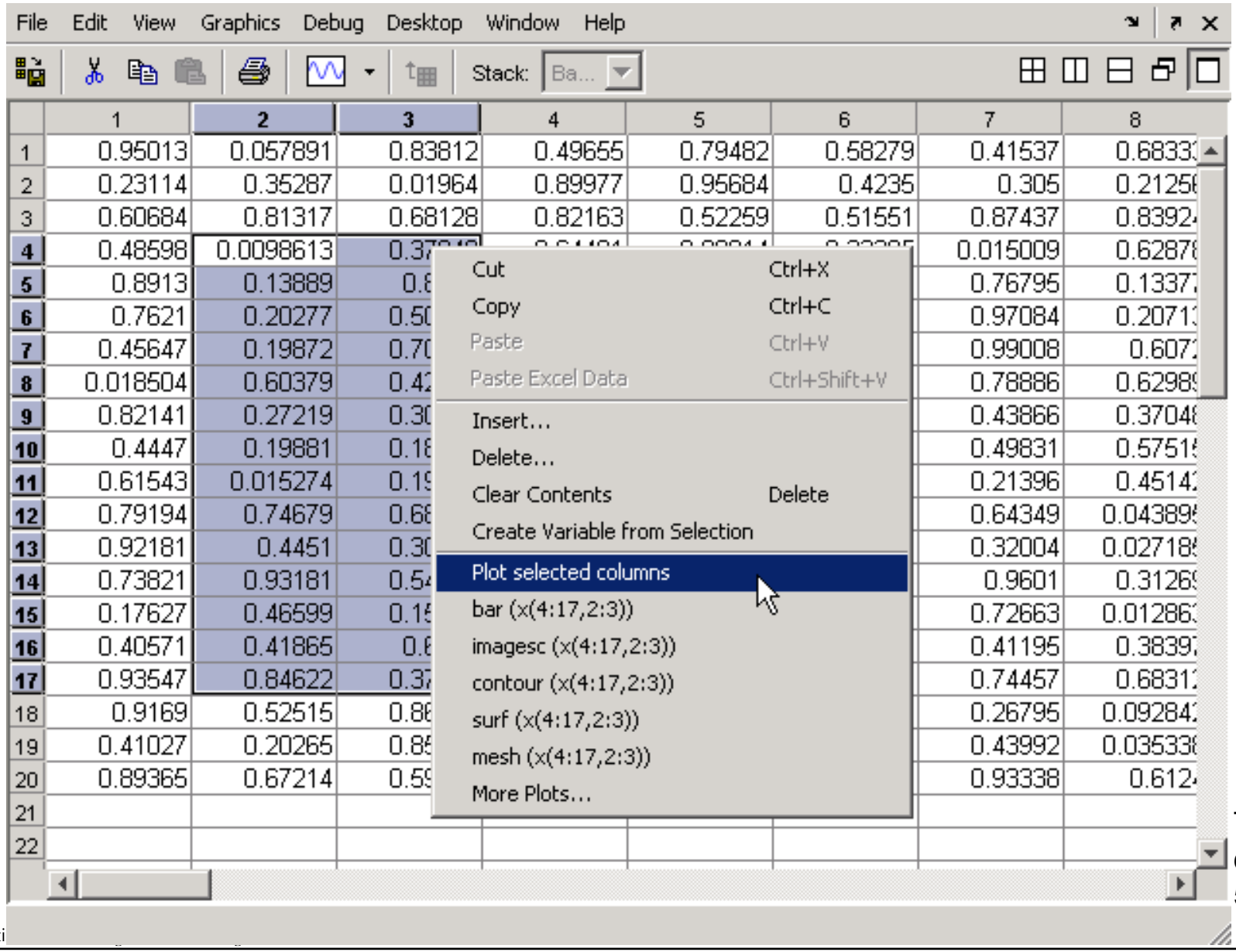
- Access to previous commands as you type



You type: PE TAB

Algunas mejoras de Matlab 7

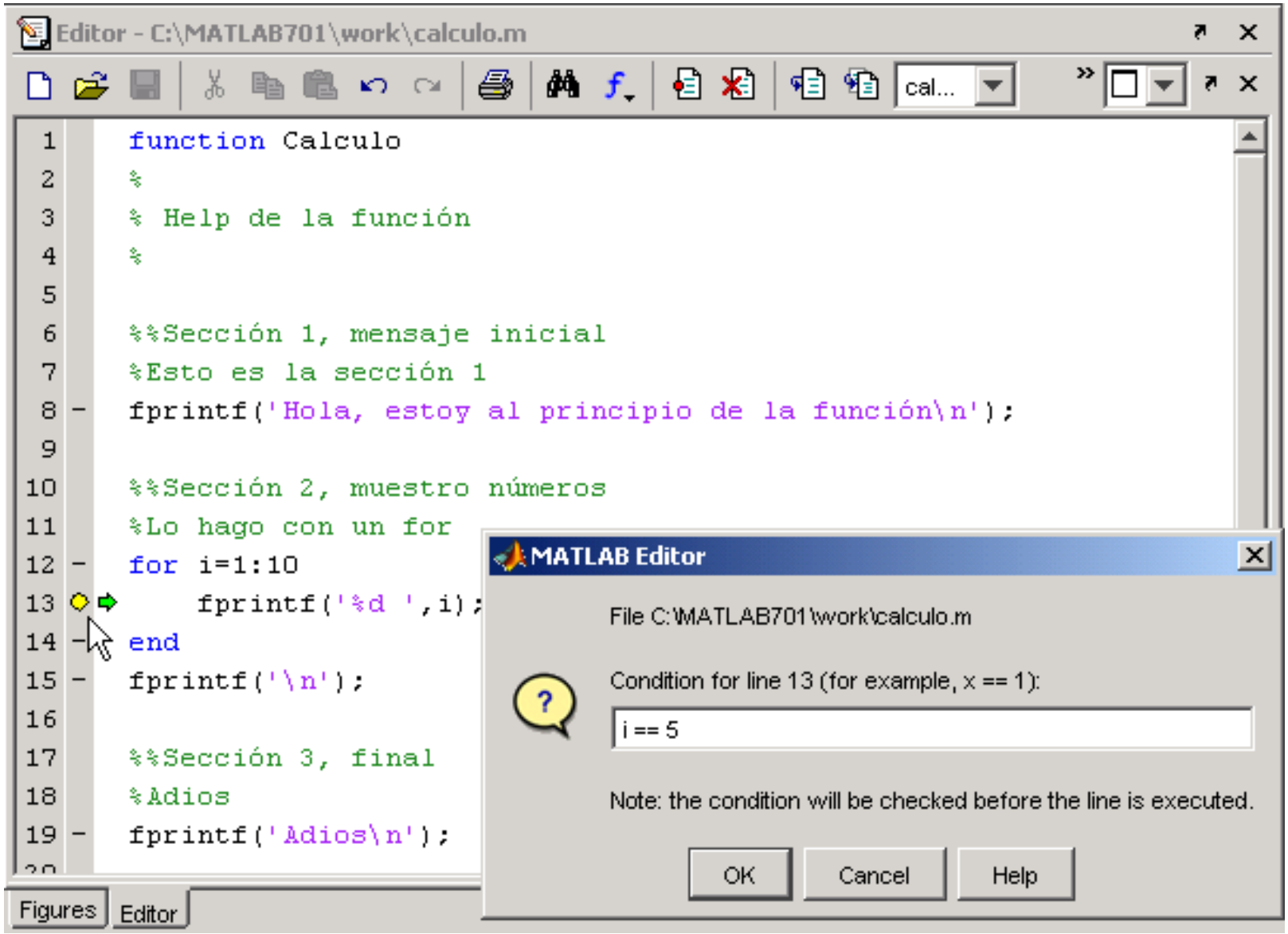
- Immediate data plot



Tamaño máximo de la matriz: 524288 elementos

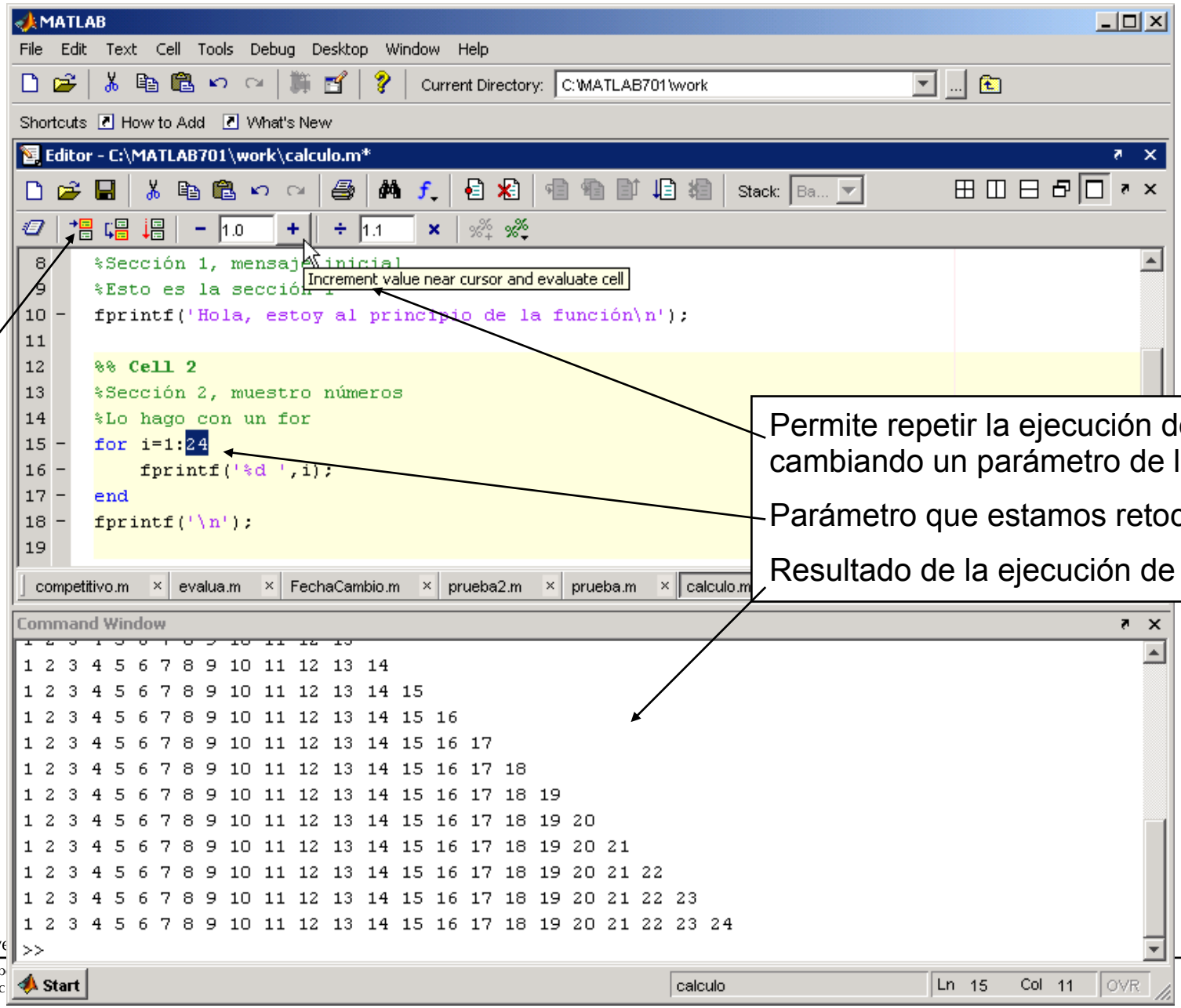
Matlab 7

- Debugger with conditional Breakpoints



Matlab 7

- Execution by sections of code (cell→enable cell mode)



Ejecución sección por sección

Permite repetir la ejecución de una sección cambiando un parámetro de la misma.

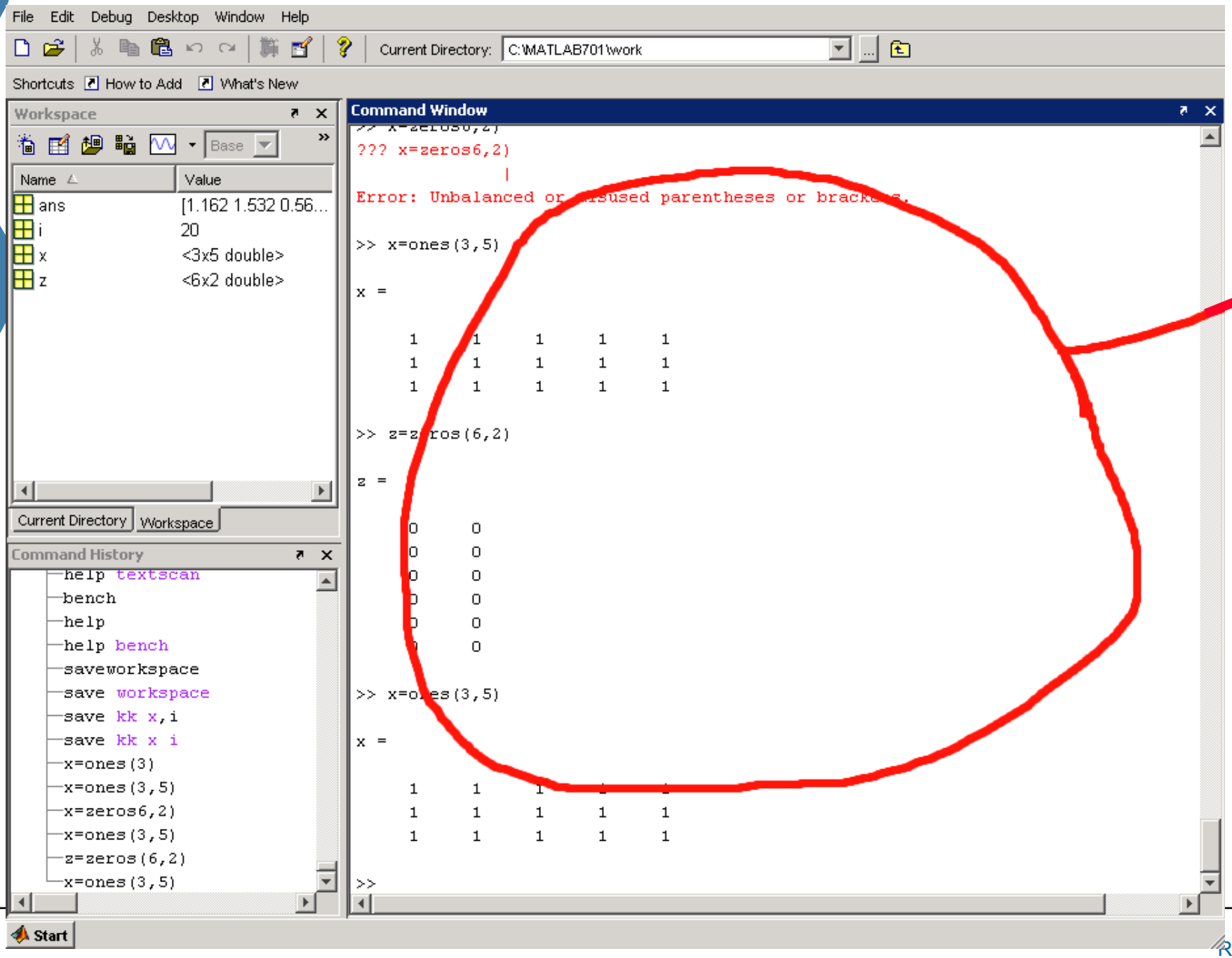
Parámetro que estamos retocando

Resultado de la ejecución de la sección

Matlab 7

- Execution/Debugger by sections of código (Cell Mode)
- Automatic Documentation generation
- Code optimization with M-Lint and profiler (see chap6)
- Better automatic code generation from Simulink
- Interactive plot tool (see chap6)
- Function `textscan` to read text files
- Computing support for integer variables
  - Smaller matrices, faster code → very useful for image processing
- Editor with international characters support

Development environment



Command window

Basic commands

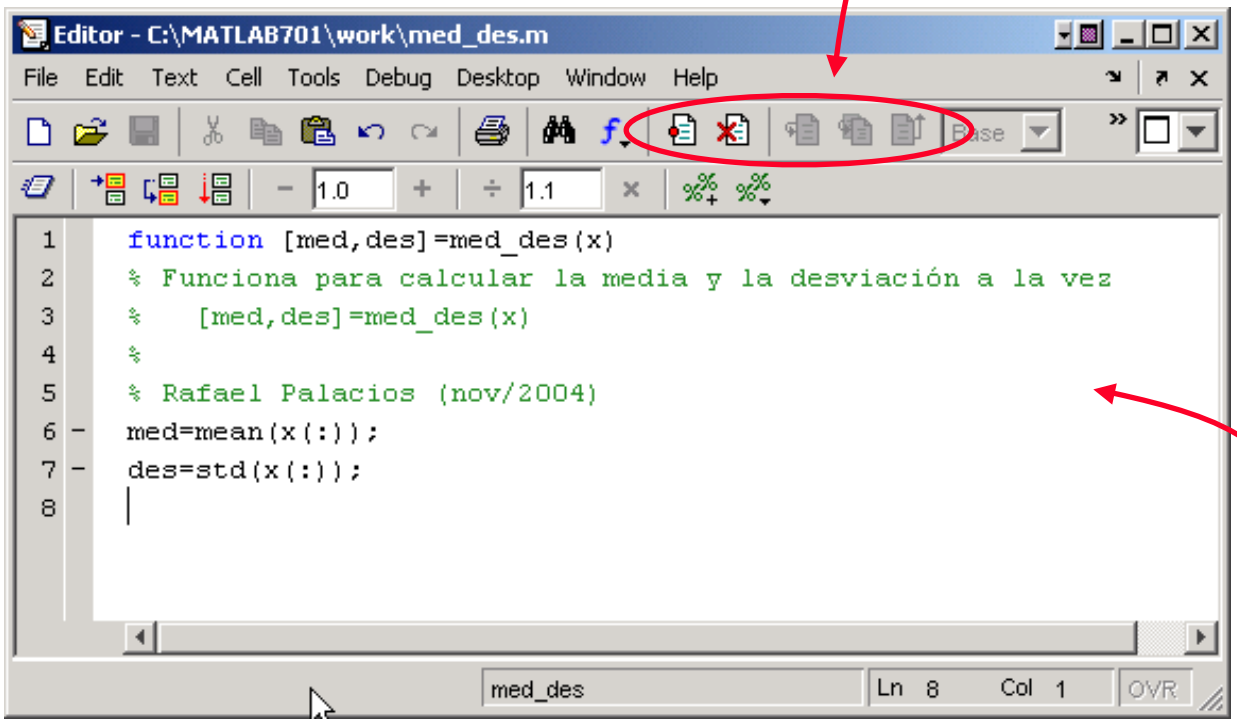
- `ver` →version number, license code and toolbox versions
  - License 46431: Research
  - License 205966: Only for teaching
- `whos` →list of variables
- `save archivo` →save all variables
- `save archivo a b` →save variables `a` & `b`
- `load archivo` → load file and create variables
- `quit` →exit

Editor

- Matlab includes and editor for writing programs and functions

Controles del debugger

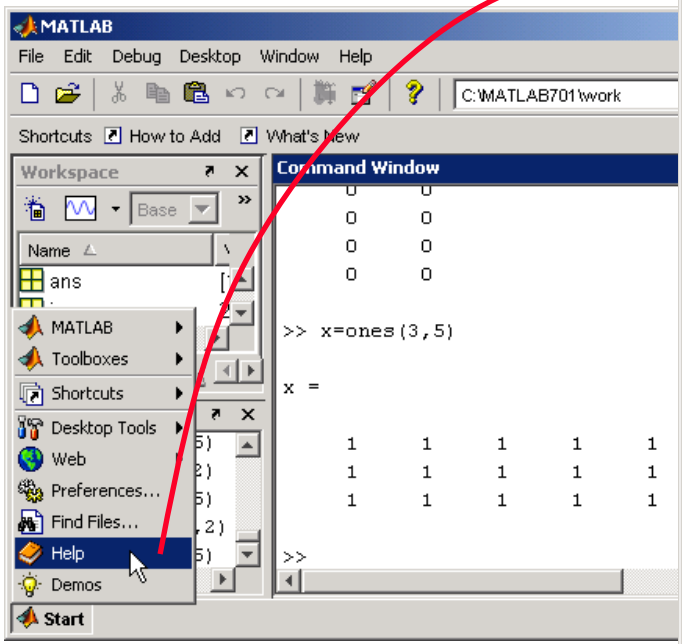
Ejecución por secciones en cell mode



sintaxis

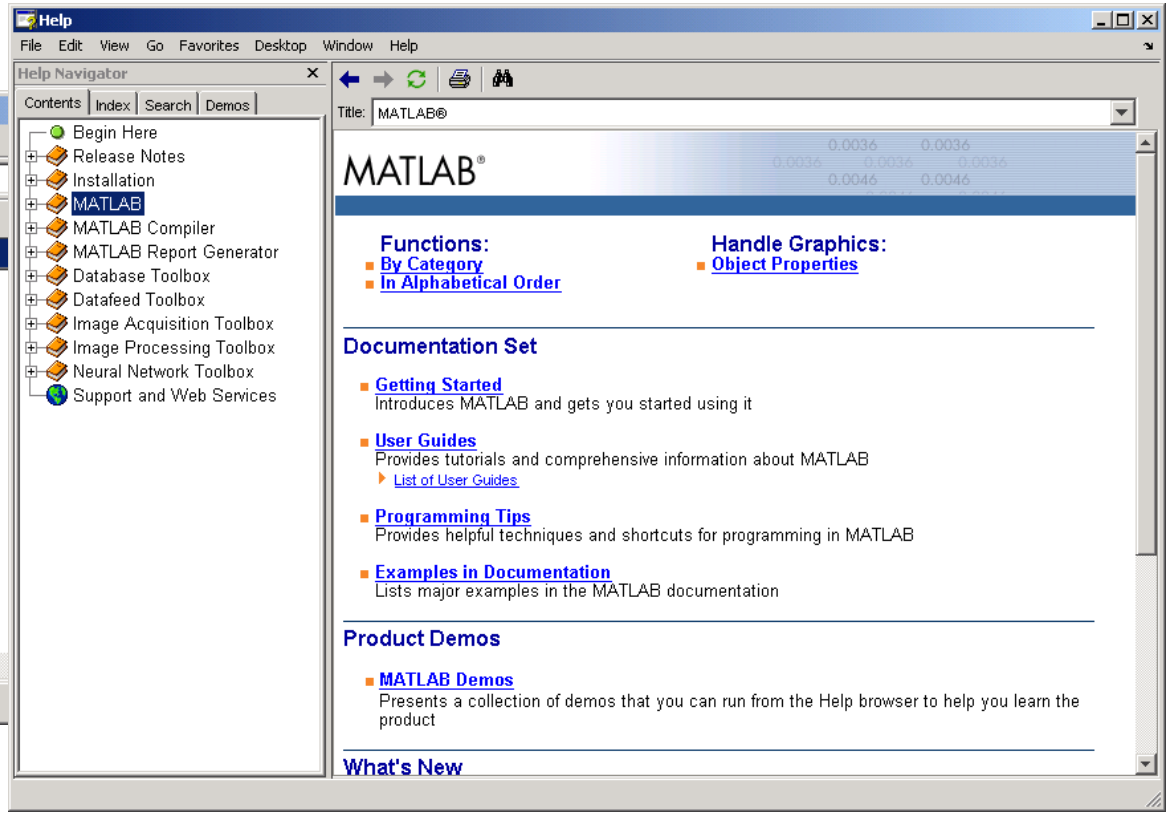
Documentation

- Help in text mode using command line
  - `help function_name` →displays help for that function
  - `help` can be used with Matlab standard functions and user developed functions
- Help in graphics mode



Start / Help

Hypertext Help Window



Documentation example

Standard sections

- Syntax
- Description
- Arguments
- Examples
- Algorithm
- Limitations
- See Also
- Refences

developed by scholars

$$f(x) = 100(x_2 - x_1^2)^2 + (a - x_1)^2$$

This changes the location of the minimum to the point `[a,a^2]`. To minimize this function for a specific value of `a`, for example `a = sqrt(2)`, create a one-argument anonymous function that captures the value of `a`.

```
a = sqrt(2);
banana = @(x)100*(x(2)-x(1)^2)^2+(a-x(1))^2;
```

Then the statement

```
[x,fval] = fminsearch(banana, [-1.2, 1], ...
   optimset('TolX',1e-8));
```

seeks the minimum `[sqrt(2), 2]` to an accuracy higher than the default on `x`.

Algorithm

fminsearch uses the simplex search method of [1]. This is a direct search method that does not use numerical or analytic gradients.

If `n` is the length of `x`, a simplex in `n`-dimensional space is characterized by the `n+1` distinct vectors that are its vertices. In two-space, a simplex is a triangle; in three-space, it is a pyramid. At each step of the search, a new point in or near the current simplex is generated. The function value at the new point is compared with the function's values at the vertices of the simplex and, usually, one of the vertices is replaced by the new point, giving a new simplex. This step is repeated until the diameter of the simplex is less than the specified tolerance.

Limitations

fminsearch can often handle discontinuity, particularly if it does not occur near the solution. fminsearch may only give local solutions.

fminsearch only minimizes over the real numbers, that is, x must only consist of real numbers and $f(x)$ must only return real numbers. When x has complex variables, they must be split into real and imaginary parts.

See Also

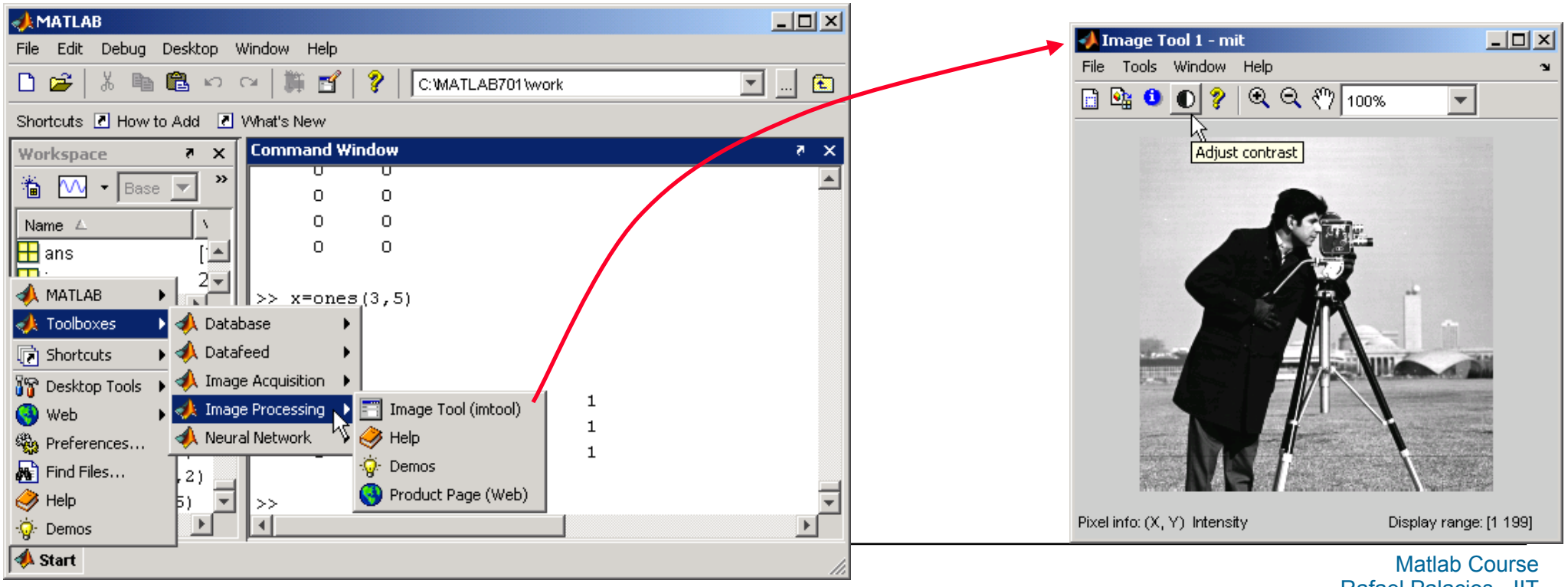
`fminbnd`, `optimset`, `function_handle` (@), anonymous functions

References

[1] Lagarias, J.C., J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," *SIAM Journal of Optimization*, Vol. 9 Number 1, pp. 112-147, 1998.

Toolboxes

- Specific libraries for different scientific topics. Include:
  - User's Guide [HTML, PDF]
  - Reference Guide [HTML, PDF]
  - Demo Programs
  - Demo applictions (called tool) ready to use



Examples of Toolboxes

```
>> ver
-----------------------------------------------------------------------------
MATLAB Version 6.5.0.180913a (R13)
MATLAB License Number: 46431
Operating System: SunOS 5.8 Generic_108528-29 sun4u
Java VM Version: Java 1.3.1_02 with Sun Microsystems Inc. Java HotSpot(TM) Server VM
-----------------------------------------------------------------------------
MATLAB                                          Version 6.5         (R13)
Simulink                                        Version 5.0         (R13)
Control System Toolbox                          Version 5.2         (R13)
Fuzzy Logic Toolbox                             Version 2.1.2       (R13)
Image Processing Toolbox                        Version 3.2         (R13)
MATLAB Compiler                                 Version 3.0         (R13)
MATLAB Web Server                               Version 1.2.2       (R13)
Mu-Analysis and Synthesis Toolbox               Version 3.0.7       (R13)
Neural Network Toolbox                          Version 4.0.2       (R13)
Nonlinear Control Design Blockset               Version 1.1.6       (R13)
Optimization Toolbox                            Version 2.2         (R13)
Real-Time Workshop                              Version 5.0         (R13)
Robust Control Toolbox                          Version 2.0.9       (R13)
SB2SL (converts SystemBuild to Simu...          Version 2.5         (R13)
Signal Processing Toolbox                       Version 6.0         (R13)
System Identification Toolbox                   Version 5.0.2       (R13)
>> date

ans =

27-Nov-2004
```

Contents

1. Introduction to Matlab.
2. Basic data types.
3. Matlab programming.
4. Advanced data types.
5. Code Optimization.
6. Graphical representation.
7. Developing standalone applications.

Variables

- Matlab doesn't require to declare variable or to specify the size of arrays
  - Variables are auto-declared when assigned
  - Memory is reallocated automatically

```
>> x=5;
>> y=20;
>> z=x*y

z =

   100

>> a=load('data.txt');
>> st='hola';
```

using ';' the value is assigned but the result is not diplayed

without ';' the final result is shown

Vectors and Matrices

- Matlab considers that all variables are matrices
- Vectores and scalars are special cases (size 1).

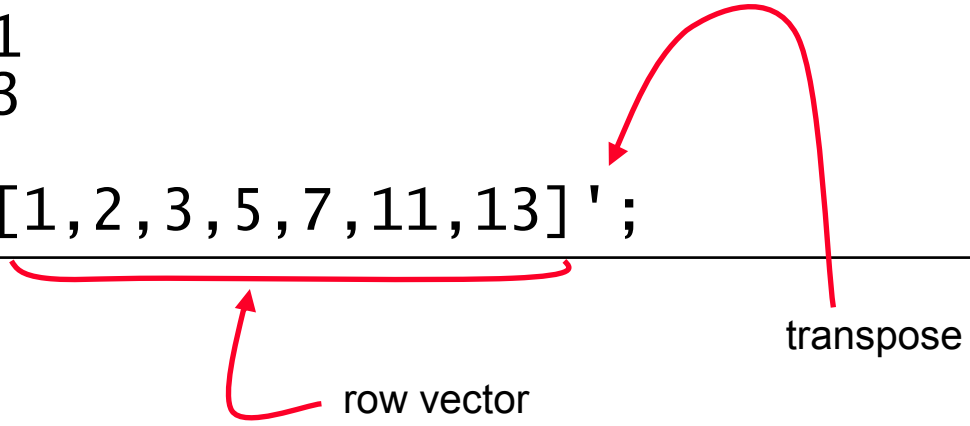
Some examples on how to initialize **row vectors**

```
>> x=[1,2,3,5,7,11,13];  → [ 1  2  3  5  7  11  13 ]
>> x=[1 2 3 5 7 11 13];  → [ 1  2  3  5  7  11  13 ]

>> y=1:5;                → [ 1  2  3  4  5 ]
>> even=2:2:10;          → [ 2  4  6  8  10 ]
>> odd_down=9:-2:1;      → [ 9  7  5  3  1 ]

>>a(5)=7;                 → [ 0  0  0  0  7 ]
```

Vectors and Matrices

Some examples on how to initialize **column vectors**

```
>> x=[1;2;3;5;7;11;13]

x =

     1
     2
     3
     5
     7
    11
    13

>> x=[1,2,3,5,7,11,13]';
```



Vectors and Matrices

Examples on how to initialize matrices

```
>> M = [1 2 3; 4 5 6; 7 8 9];

>> ceros=zeros(2,5);

>> unos=ones(3,4);

>> M2=[ 20, 21, 22;  M];
>> M2=[[20, 21, 22]; M];

>> M3=[ [15;16;17], M];

>> aleatorio=rand(20,30);
>> normal=randn(20,30);
```

```
1     2     3
4     5     6
7     8     9
```

```
0     0     0     0     0
0     0     0     0     0
```

```
1     1     1     1
1     1     1     1
1     1     1     1
```

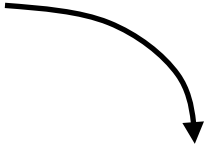
```
20    21    22
 1     2     3
 4     5     6
 7     8     9
```

```
15     1     2     3
16     4     5     6
17     7     8     9
```

Accessing elements of a matrix

- Matlab uses parenthesis ( ) to access elements of a matrix
- Indexes start at 1, so the first of matrix mat is `mat(1,1)`
- Example: `a(3,5)=56.8;`

```
0.1737    0.3421    0.6391    0.1632    0.2313
0.7858    0.7742    0.0934    0.2763    0.8453
0.3656    0.1478    0.9288    0.1310    0.7264
0.7769    0.1482    0.4851    0.0232    0.6947
```

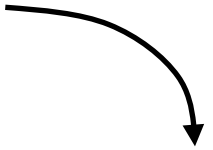


```
0.1737    0.3421    0.6391    0.1632    0.2313
0.7858    0.7742    0.0934    0.2763    0.8453
0.3656    0.1478    0.9288    0.1310   56.8000
0.7769    0.1482    0.4851    0.0232    0.6947
```

Accessing elements of a matrix

- Vectors can be used to specify the indexes
- Example 1: `a(2:3,1:4)=zeros(2,4);`
  also: `a(2:3,1:4)=0;`

0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

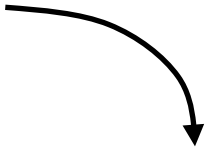


0.1737	0.3421	0.6391	0.1632	0.2313
0	0	0	0	0.8453
0	0	0	0	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

Accessing elements of a matrix

- Ejemplo 2: `a([2,3],[2,4])=ones(2,2);`
  o bien: `a([2,3],[2,4])=1;`

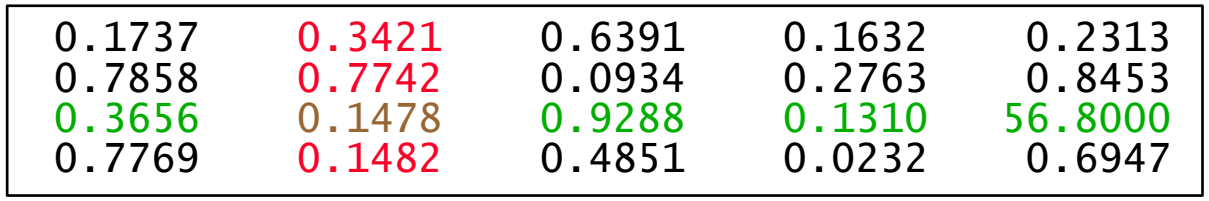
```
0.1737    0.3421    0.6391    0.1632    0.2313
     0         0         0         0    0.8453
     0         0         0         0   56.8000
0.7769    0.1482    0.4851    0.0232    0.6947
```



```
0.1737    0.3421    0.6391    0.1632    0.2313
     0    1.0000         0    1.0000    0.8453
     0    1.0000         0    1.0000   56.8000
0.7769    0.1482    0.4851    0.0232    0.6947
```

Accessing elements of a matrix

- The symbol ' : ' means “all elements”
- It may be used as “all elements in row” or “all elements in the matrix”, etc.



a(3,:)
size(a(3,:)) → [1 5]

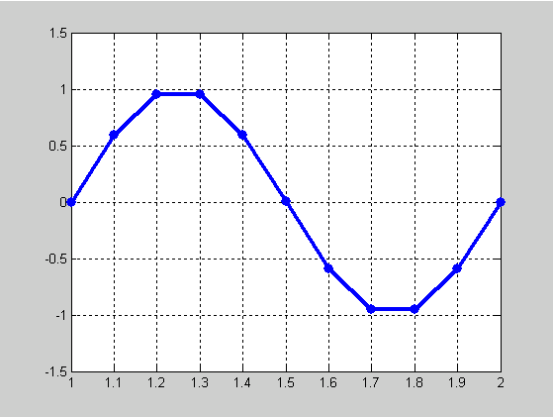
a(:,2)
size(a(:,2)) → [4 1]

a(:) → all elements
size(a(:)) → [20 1]
returns a column vector

Acceso a los elementos de una matriz

- The operator 'end' means “last element"
  - Example: Vector of differences

```
>> t=1:0.1:2;
>> y=sin(2*pi*t);
>> differences=[NaN; y(2:end)-y(1:end-1)];
```




```
t =
    1.00    1.10    1.20    1.30    1.40    1.50    1.60    1.70    1.80    1.90    2.00

y =
   -0.00    0.59    0.95    0.95    0.59    0.00   -0.59   -0.95   -0.95   -0.59   -0.00

differences =
      NaN    0.59    0.36    0.00   -0.36   -0.59   -0.59   -0.36   -0.00    0.36    0.59
```

Basic mathematical operators

- Arithmetic operators: + - * / ^
  - Matlab works with matrices, in contrast with other programming languages that only work with scalar values

```
>> a=rand(2,5);
>> b=rand(5,2);
>> c=a*b;                % matrix 2x2
>> d=b*a;                % matrix 5x5
```

- Operations element-by-element: + - .* ./ .^

```
mn = magic(4);
```

```
 16     2     3    13
  5    11    10     8
  9     7     6    12
  4    14    15     1
```

mn*mn
mn^2

```
345   257   281   273
257   313   305   281
281   305   313   257
273   281   257   345
```

mn.*mn
mn.^2

```
256     4     9   169
 25   121   100    64
 81    49    36   144
 16   196   225     1
```

Other matrix operations

- Sum: `sum`

```
>> b=sum(A);   % if A is a matrix, add elements by column. b is a row vector
>> c=sum(b);   % if b es vector, all elements are added. c is a scalar

>> c=sum(sum(a));  % addition of all elements in a
>> c=sum(a(:));    % addition of all elements in a
```

- Mean and stander deviation: `m=mean(A);` `sigma=std(A);`
- Elements of the diagonal: `v=diag(A);`
- Left division: `x=A\B;` The least squares solution for A x = b is obtained by means of x = A \ b;
- Determinant: `c=det(A);`
- Inverse: `B=inv(A);`
- Eigenvalues: `v=ein(A);`
- Absolute values, or complex module: `B=abs(A);`

Other functions

- Trig: `sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh`
- Rounding: `floor, ceil, round, fix`
- Modular: `rem, mod`
- Exponential: `exp, log, log2, log10, sqrt`
- Primes: `factor, primes`

Data types

- Matlab uses type **double** according to the IEEE standard
- It can manage special values like inf (infinity) y NaN (not-a-number)
- Complex number are used automatically when needed.

```
>> a=123/0
Warning: Divide by zero.
a =
   Inf

>> b=0/0
Warning: Divide by zero.
b =
   NaN

>> Inf-Inf
ans =
   NaN

>> c=15+sqrt(-1)
c =
  15.0000 + 1.0000i
```

Data types

- Real matrices
  - double `realmin→2.2251e-308, realmax→1.7977e+308, eps→2.2204e-016`
  - single `realmin→1.1755e-038, realmax→3.4028e+038, eps→1.1921e-007`
- Integer matrices
  - int8, uint8
  - int16, uint16
  - int32, uint32
  - int64, uint64
- Other types
  - char
  - logical
  - cell
  - struct

Data types: sparse matrices

- Sparse matrices stored as sparse safe memory and computer faster

```
s = sparse(1000,1000);
s(2,11) = 2;
s(992,875) = 3;
s(875,992) = 4.7;
```

- All mathematical operations are executed using sparse algorithms.
- When they are not really sparse, they are automatically converted back to normal matrices

```
s=s+3;   % s is not sparse anymore
```

Contents

1. Introduction to Matlab.
2. Basic data types.
3. Matlab programming.
4. Advanced data types.
5. Code Optimization.
6. Graphical representation.
7. Developing standalone applications.

Scripts

- One **script** is a sequence of instructions that matlab can save in a `.m file`

```
%Script de ejemplo

%% Inicio
a=magic(4);
fprintf('Inicio cálculos\n');

%% Traza
traza=sum(diag(a));

%% Resultado
fprintf('La traza vale: %f\n',traza)
```

`ejem_script.m`

- It is executed by the filename:

```
>> ejem_script
```

Functions (Call)

- Matlab functions can receive several arguments and they can also return several results:
  [m,d]=med_des(x);

- Functions can have optional arguments

```
mit=imread('cameraman.tif','TIFF');
mit=imread('cameraman.tif');
```

- It is not necessary to assign all results

```
[mit,map]=imread('imageman.gif');
mit=imread('imageman.gif');
```

Functions (declaration)

- Functions are also written in `.m` files that must me located in the current working directory (or a directory specified in the `path`)

```
function [med,des]=med_des(x)
% Funciona para calcular la media y la desviación a la vez
%   [med,des]=med_des(x)
%
% Rafael Palacios (nov/2004)
med=mean(x(:));
des=std(x(:));
```

`Med_des.m`

This is the information that you get if you type: `help med_des`

Functions

- The variable `nargin` (local to the function) holds the number of arguments received.
- The variable `nargout` (local to the function) holds the numbers of results that will be used in the calling functions. Therefore we can skip some computing sections.
- All arguments work by value, not by reference. (but it is smart to save time and memory)

> •Scripts share variables with the workspace, while functions use local variables

Logical expressions

- Relational operators: `~= == > < >= <=`
- Logical operators:

&&	Short-circuit AND
||	Short-circuit OR
&	AND
|	OR

- There is a function for `xor`, but not an operator

Control de Flujo: if

- if

```
if a > b
   tmp=a;
   a=b;
   b=tmp;
end
```

```
if rem(n,2) ~= 0
   M = odd_magic(n)
elseif rem(n,4) ~= 0
   M = single_even_magic(n)
else
   M = double_even_magic(n)
end
```

In contrast with C, Matlab does not requiere parenthesis for the logical expression

Control de Flujo: for

- `for` loop

```
for n = 3:32
   r(n) = rank(magic(n));
end
```

```
a=[];
for n = [ 1 2 3 5 7 11 ]
   a = [a, isprime(n)];
end
```

Control de Flujo: while

- while loop

```
while ~isprime(x)
    x = x + 1;
end
```

Control de Flujo: switch

- switch-case

```
switch (rem(n,4)==0)+(rem(n,2)==0)
    case 0
       M = odd_magic(n)
    case 1
       M = single_even_magic(n)
    case 2
       M = double_even_magic(n)
    otherwise
       error('This is impossible')
 end
```

In contrast with C, Matlab doesn't use `break`.

Control de Flujo: try

- `try-catch`

```
try
   statement
   ...
   statement
catch
   statement
   ...
   statement
end
```

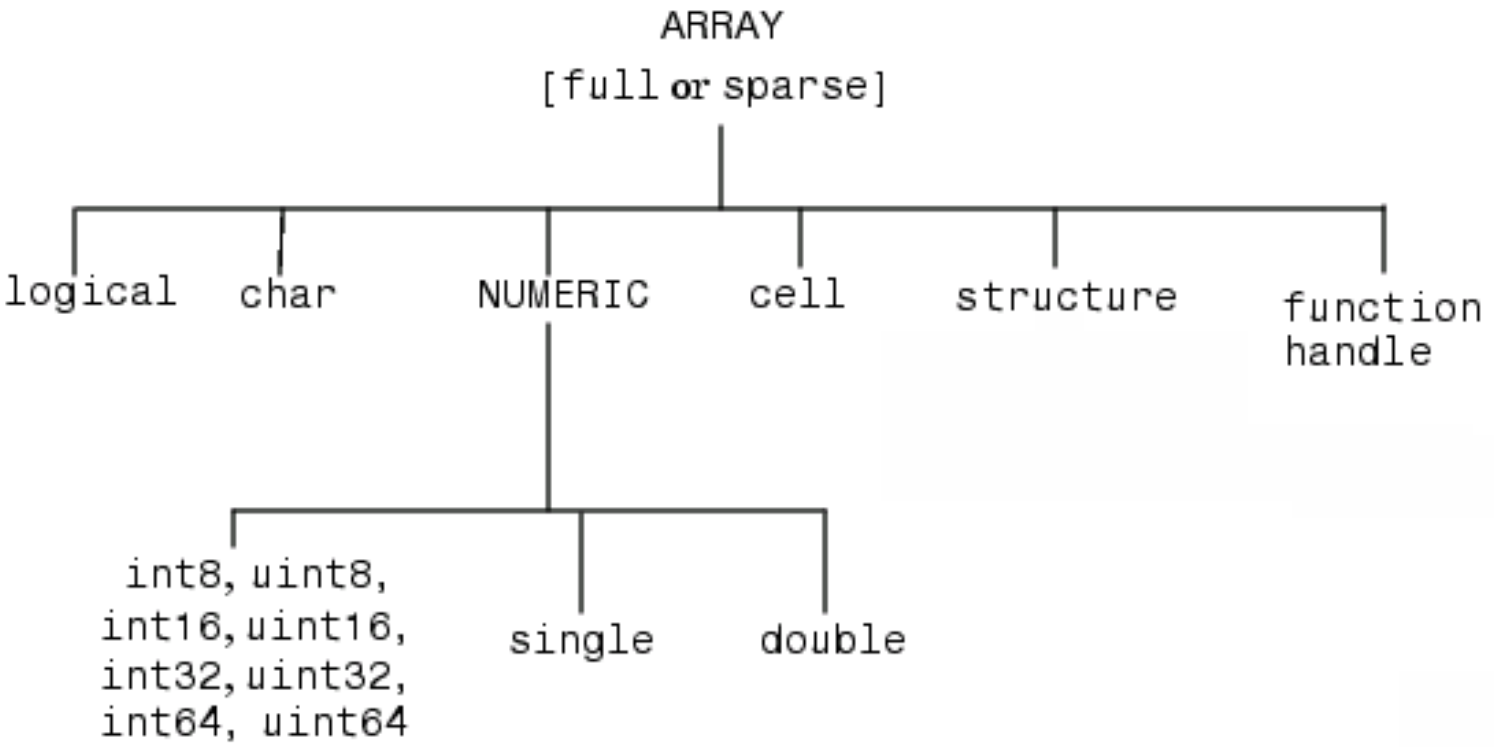
The instructions between `catch` and `end` are only executed in case of error among the first set of statements. One may use `lasterr` to get the error code that triggered the catch section.

Contents

1. Introduction to Matlab.
2. Basic data types.
3. Matlab programming.
4. Advanced data types.
5. Code Optimization.
6. Graphical representation.
7. Developing standalone applications.

All data types

- Matlab has 15 data types that can be used to build matrices or arrays



In addition there are user defined data types for object oriented programming: *user classes*, y *Java classes*

Identifying the type of data

- Decription of the data type

```
>> type=class(x)
type =
double
>>
```

- Logica identification

```
isinteger(x)
isfloat(x)
ischar(x)
islogical(x)
iscell(x)
isstruct(fecha)
```

```
int8, uint8
int16, uint16
int32, uint32
int64, uint64
```

```
single
double
```

```
isempty([])
isinf(Inf)
isnan(NaN)
```

Type conversion

- Convertion uses the name of the data type as a function name

```
>> a=randn(5,7)

a =

   -0.4326    1.1909   -0.1867    0.1139    0.2944    0.8580   -0.3999
   -1.6656    1.1892    0.7258    1.0668   -1.3362    1.2540    0.6900
    0.1253   -0.0376   -0.5883    0.0593    0.7143   -1.5937    0.8156
    0.2877    0.3273    2.1832   -0.0956    1.6236   -1.4410    0.7119
   -1.1465    0.1746   -0.1364   -0.8323   -0.6918    0.5711    1.2902

>> b=int8(a)

b =

    0    1    0    0    0    1    0
   -2    1    1    1   -1    1    1
    0    0   -1    0    1   -2    1
    0    0    2    0    2   -1    1
   -1    0    0   -1   -1    1    1
```

Matlab applies rounding while converting to integer

Type conversion

- Rounding functions
  - `round`: nearest integer
  - `floor`: rounds towards -Inf
  - `ceil`: rounds towards +Inf
  - `fix`: rounds towards 0

```
>> class(round(3.5))
ans =
double


>> b(4.7)
??? Subscript indices must either be real positive integers or logicals.

>> b(round(4.7))
ans =
   -1
```

Does not change the data type

Round is useful for indexing with real values

Character strings

- In matlab strings are vectors of character (like in C)

```
>> str='Hello World';
>> whos
  Name      Size                     Bytes  Class
  str       1x11                        22  char array
Grand total is 11 elements using 22 bytes

>> str(7)
ans =
W

>> str=['H', 'o', 'l', 'a'];

>> nombres(1,:)='Rafael';
>> nombres(2,:)='Ana   ';
```

Variables that contain several names, build a matrix of char in which all the names have the same length. The conversion function `char` helps to build such matrix

```
>> names=char('Rafael', 'Ana');
```

Using **cell arrays** it is possible to store several strings of different lengths into one variables

Strings

- Comparing two strings
  - The operator == works in vector mode, element-by-element

```
A = 'fate';
B = 'cake';

A == B
ans =
     0     1     0     1
```

  - `strcmp`: compares strings and returns 1 if they are equal and 0 if differents. (Note that is different that C).
  - Other comparing functions: `strncmp`, `strcmpi`, `strncmpi`

Strings

- Other functions for strings
  - `strrep`: typical find-and-replace

```
st2=strrep(st1,'find','replace');
```

  - `findstr`: find a string within another

```
pos = findstr('find', st);
```

  - `strcat`: concat 2 or more strings

```
text = strcat(st1, st2, st3);
```

  - `sprintf`: builds a string. Equivalent to `sprintf` in C

```
st=sprintf('I have %6.2f EUR',my_money);
```

Structures and Cell Arrays

- Structures allows matlab to store several variables of different types under one variable name

```
>> dot.x=123;
>> dot.y=34;
>> dot.color='red';
>> dot
dot =
         x: 123
         y: 34
     color: 'red'
```

  - Structures do not requiere previous definition
  - Fields access is similar to C
  - It is possible to use vectors of structures

```
>> punto(2).x=435;
```

Structures and Cell Arrays

- It is possible to use variables to specify the field name:

```
>> punto.x=123;
>> punto.y=34;
>> punto.color='red';
>> punto
punto =
        x: 123
        y: 34
    color: 'red'

>> campo='color';
>> punto.(campo)
ans =
red
```

Structures and Cell Arrays

- A cell array is used to build vectors in which each element may use a different data type:

```
>> c={12,'Red',magic(4)};
>> c
c =
    [12]    'Red'    [4x4 double]


>> b{1}=12;
>> b{2}='Red';
>> b{3}=magic(4);
>> b
b =
    [12]    'Red'    [4x4 double]
```

  - One should use { } instead of [ ] or ( )
  - The difference with structures is that you may use indexes instead of field names
  - Structures and cell arrays are less efficient that matrices

Structures and Cell Arrays

- Cell array can be used to build a matrix in which each element ha a different type:

```
>> a{1,1} = 12;
>> a{1,2} = 'Red';
>> a{1,3} = magic(4);
>> a{2,1}=ones(3);
>> a{2,2}=43;
>> a{2,3}='texto';
>> a
a =
    [         12]    'Red'    [4x4 double]
    [3x3 double]    [ 43]    'texto'
>>
```

is this useful?

Structures and Cell Arrays

- Using ( ) we can access the element, which is type cell
- Using { } we can access the value directly

```
>> a
a =
    [         12]    'Red'    [4x4 double]
    [3x3 double]    [ 43]    'texto'
>> class(a)
ans =
cell

>> class(a(1,1))
ans =
cell

>> class(a{1,1})
ans =
double
```

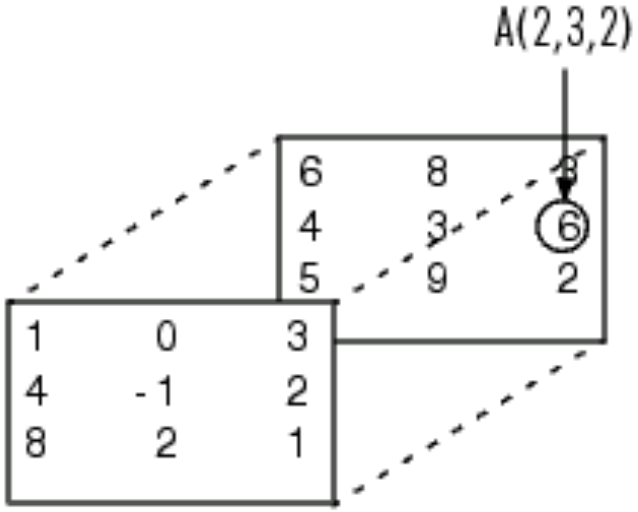
Structures and Cell Arrays

- Database access example (database toolbox)

```
conn=database('base_de_datos_ODBC','usuario','password');
query='SELECT to_number(PROD),HORA,EST FROM TB_CENT WHERE CENTRAL=''ROBLA'' ';
curs=exec(conn,query);
curs=fetch(curs);

for i=1:size(curs.Data,1)
   producciones(i)=curs.Data{i,1};
   horas(i)=curs.Data{i,2};
   estados(i)=curs.Data{i,3};
end

close(curs);
close(conn);
```

Multi-dimensional matrix

- Matrices of more that 2 dimensions are called **Multidimensional Arrays**
- Matlab support all matrix operations in N dimensiones




```
A(:,:,1) =

     1     0     3
     4    -1     2
     8     2     1

A(:,:,2) =

     6     8     3
     4     3     6
     5     9     2
```

Multi-dimensional matrix

```
>> c=imread('autumn.tif');
>> whos c
  Name        Size                          Bytes  Class
  c         206x345x3                      213210  uint8 array

Grand total is 213210 elements using 213210 bytes

>> imshow(c)

>> max(c(:))

ans =

   248

>> gris=(c(:,:,1)+c(:,:,2)+c(:,:,3))/3;

>> imshow(gris)
```

todos los elementos

Date and Time

- Matlab can use dates and times in three formats:
  - strings
  - numeric value (number of days since 1/ene/0000)
  - numeric vector [year, month, day, hour, min, sec]

Date Format	Example	
Date string	02-Oct-1996	date
Serial date number	729300	now
Date vector	1996 10 2 0 0 0	clock

- It takes into account leap years
- It does not take into account UTC/local time or DST

Date and Time

- Date/Time onversion functions

Function	Description
datenum	Convert a date string to a serial date number.
datestr	Convert a serial date number to a date string.
datevec	Split a date number or date string into individual date elements.

you can use `fprintf`

- Examples

```
function fecha_corregida=FechaCambio(fecha_calculo,dias)
%
%Obtiene una nueva estructura de fecha adelantando o retrasando dias
%function fecha_corregida=FechaCambio(fecha_calculo,dias)
%   fecha_corregida y fecha_calculo son estructuras con los campos dia, mes, aNo.
%
%Ejemplo: function fecha_corregida=FechaCambio(fecha_calculo,-1); %dia anterior
%
%Rafael Palacios Nov/2004
%
fecha_num=datenum(fecha_calculo.aNo,fecha_calculo.mes,fecha_calculo.dia);
fecha_num=fecha_num+dias;
fecha_vec=datevec(fecha_num);
fecha_corregida.aNo=fecha_vec(1);
fecha_corregida.mes=fecha_vec(2);
fecha_corregida.dia=fecha_vec(3);
```

Contents

Measuring time

- Basic functions to measure execution time
  - `tic` & `toc` measure elapsed time in seconds

```
>> tic; inv(inv(inv(randn(1000)))); toc
Elapsed time is 10.015000 seconds.
```

```
tic
   for k = 1:100
      -- programa rápido --
   end
toc
```

  - `cputime` measures CPU time in seconds

```
>> t=cputime; inv(inv(inv(randn(1000)))); e=cputime-t
e =
    9.5137
```

Code analysis

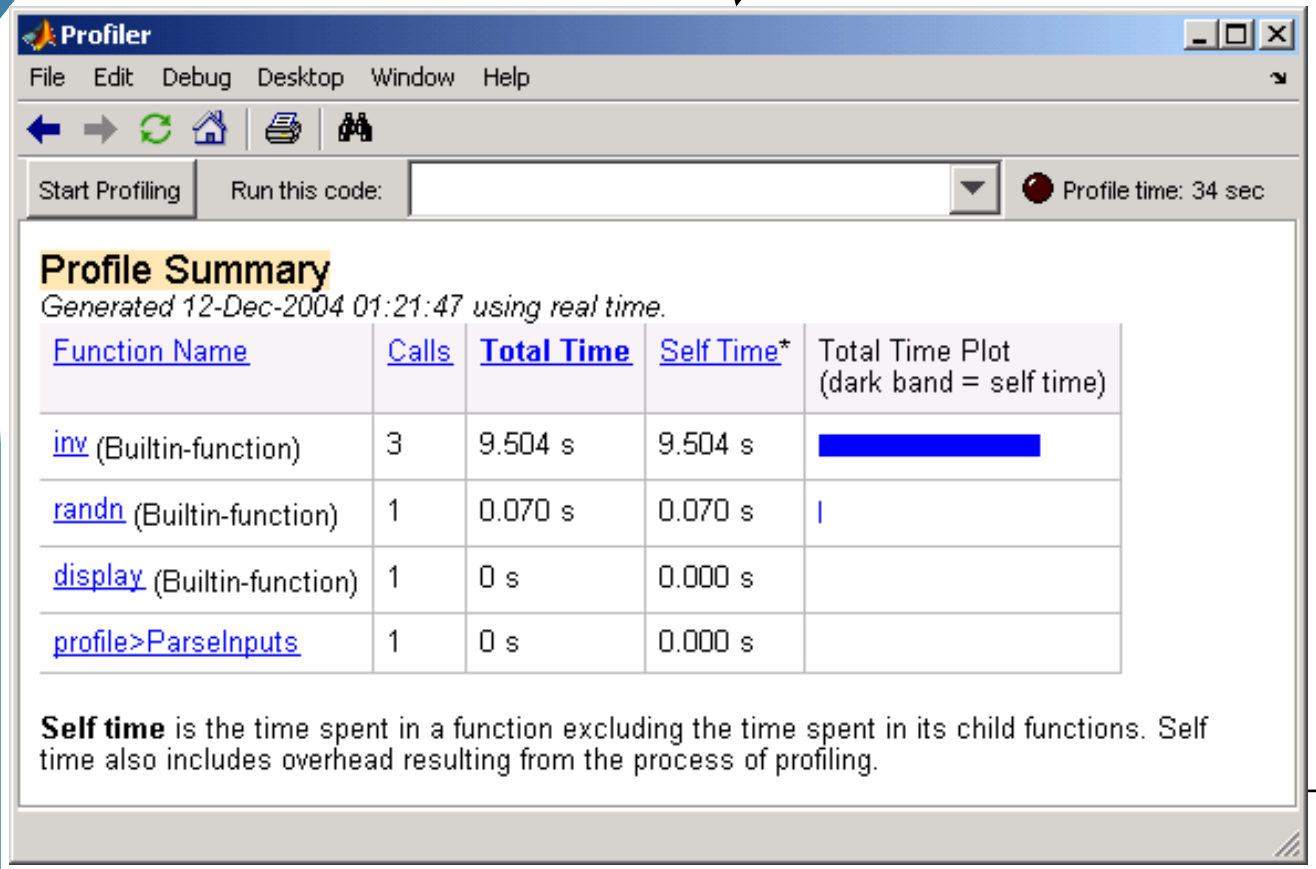
- `profiler` generates a report of the performance of the code

Graphical mode:

```
>> profile viewer
```

Commands

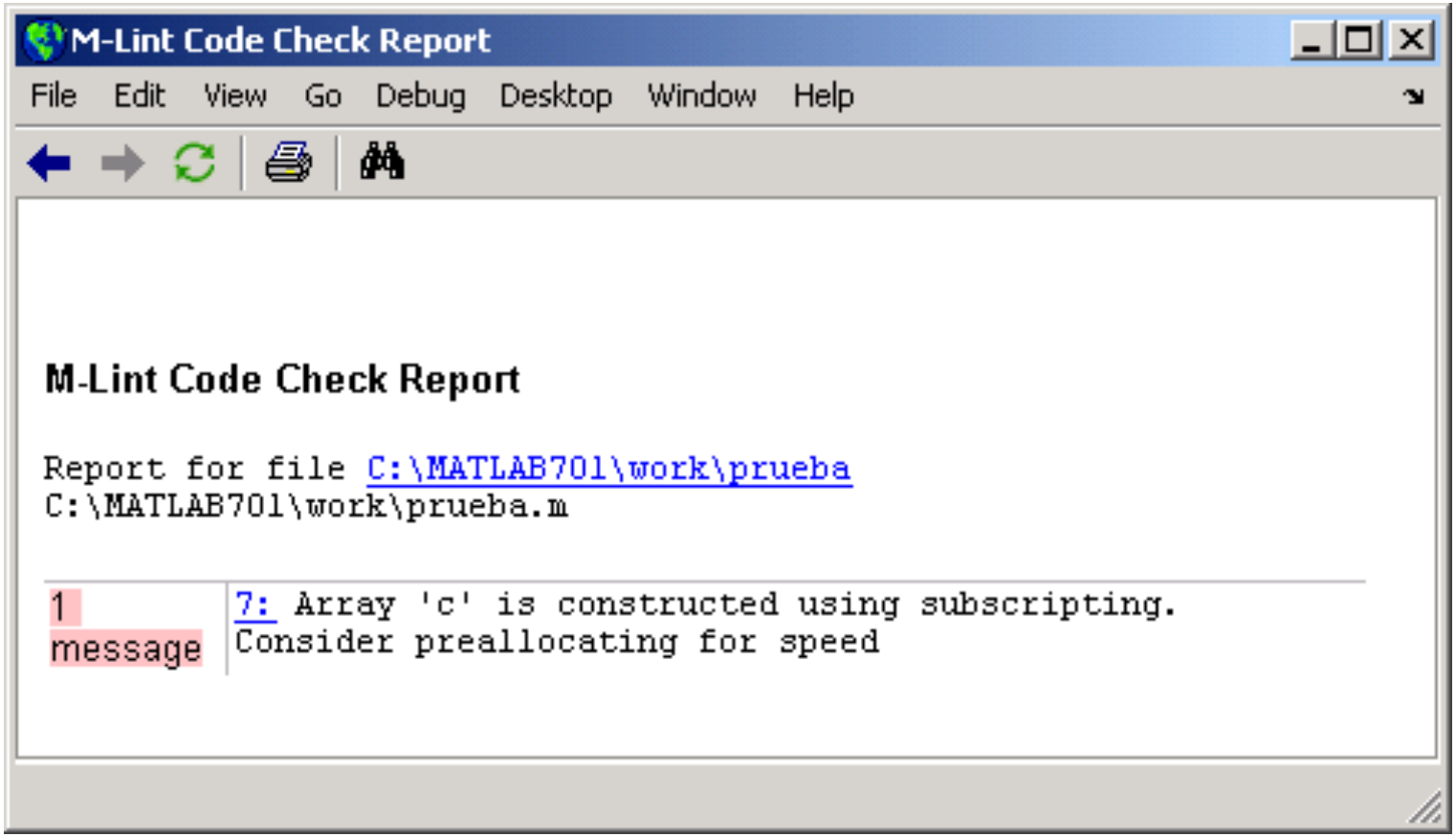
```
>> profile on
>> inv(inv(inv(randn(1000))));
>> profile off
>> profile report
```



Profiler detects the function that it is worth improving.

Code analysis

- `M-Lint` is used to analyze the code and automatically detect possible improvements.



Optimizing loops

- Loops are slow in Matlab

```
>> tic, for t=1:100, prueba, end, toc
Elapsed time is 3.856000 seconds.

>> tic, for t=1:100, prueba2, end, toc
Elapsed time is 2.554000 seconds.

>> tic, for t=1:100, suma=sum(z(:)); end, toc
Elapsed time is 1.893000 seconds.
```

```
%prueba2
suma=0;
for i=1:1000
    for j=1:1000
        suma=suma+z(j,i);
    end
end
```

```
%prueba
suma=0;
for i=1:1000
    for j=1:1000
        suma=suma+z(i,j);
    end
end
```

Preallocation

- Preallocation avoids dynamic memory allocations

500x

```
>> tic, prueba, toc
Elapsed time is 54.589000 seconds.

>> tic, for t=1:100, prueba2, end, toc
Elapsed time is 10.846000 seconds.
```

500 veces más rápido!!!!

```
%prueba2
z2=ones(size(z));
for i=1:1000
    for j=1:1000
        z2(j,i)=z(j,i);
    end
end
```

```
%prueba
for i=1:1000
    for j=1:1000
        z2(j,i)=z(j,i);
    end
end
```

Nota: Estos tiempos no mejoran utilizando el compilador porque son retrasos del sistema operativo, no de Matlab. En Unix el código malo es 50 veces más lento, en lugar de 500 veces más lento.

Use Find instead of for

- Very often it is possible to use `find` instead of a loop
  - `find` returns a vector of indexes corresponding to the values "true"

100x

Example: find pixels with values larger that 200

```
>> tic, for t=1:100, prueba, end, toc
Elapsed time is 6.0 seconds.

>> tic, for t=1:100, length(find(c>200)); end, toc
Elapsed time is 0.06 seconds.

tic, for t=1:100, z=c>200; sum(z(:)); end, toc
Elapsed time is 0.044 seconds.
```

```
%prueba imagen
num=0;
for i=1:size(c,1)
    for j=1:size(c,2)
        for k=1:size(c,3)
            if c(i,j,k)>200
                num=num+1;
            end
        end
    end
end
```

MacBook Pro: Intel Core 2 Duo 2.26 GHz

Use Find instead of for

Example2: Set to zero all pixel with a value larger that 200

```
>> c=imread('autumn.tif');
>> tic, for t=1:100, my_loop, end, toc
Elapsed time is 5.6 seconds.

>> c=imread('autumn.tif');
>> tic, for t=1:100, c(find(c>200))=0; end, toc
Elapsed time is 0.06 seconds.

>> tic, for t=1:100, c(c>200)=0; end, toc
Elapsed time is 0.05 seconds.
```

100x

my_loop

```
%prueba imagen
for i=1:size(c,1)
    for j=1:size(c,2)
        for k=1:size(c,3)
            if c(i,j,k)>200
                c(i,j,k)=0;
            end
        end
    end
end
```

c>200 generates a 3D matrix of true or false

`find(c>200)` generates a vector with the indexes of true
`c(find(c>200))` is equivalent to say `c([23, 267,...])`

`(c>200)` is a vector of the same size as c full of logical values that can be use as indexes.

MacBook Pro: Intel Core 2 Duo 2.26 GHz

Use Find instead of for

- Other useful functions similar to find:
  - `all`: determines if all elements are nonzero
    - `If all(A>0.5)`
  - `any`: determines if any element is nonzero
    - `If any(A>0.5)`
  - `reshape`: rearrage the elements of a matrix to a different matrix shape.
  - `sort`: sorts elements and obtains the indexes

```
function x=aleat(rango)
%% function x=aleat(rango)
%  Genera una lista de números aleatorios no repetidos de tamaño rango
%
z=rand(1,rango);
[s,x]=sort(z);
```

Global variables

- Functions treat variables by value
  - There are many calls of this type:
    ```
    my_date=NextDay(my_date);
    ```
  - Functions that transform matrices are very inefficient

Although variable are used by value, Matlab is smart enough to treat them by reference (more efficient) if they are not modified inside the function.

Global variables

- In general using global variables is not recommended for code clarity. However it can improve the efficiency if working with large matrices.

```
global GRAVITY
GRAVITY = 32;
y = falling((0:.1:5)');
```

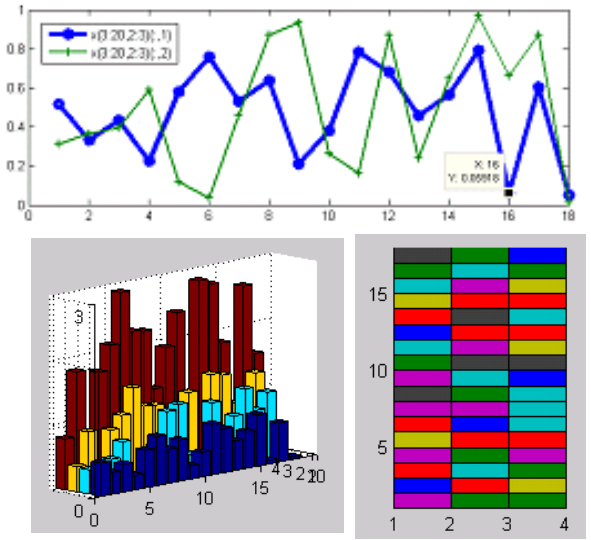
```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

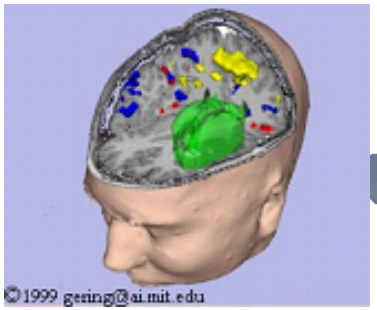
Contents

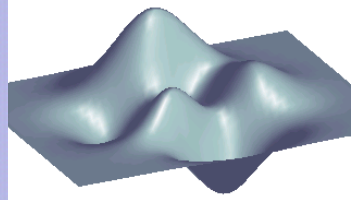
1. Introduction to Matlab.
2. Basic data types.
3. Matlab programming.
4. Advanced data types.
5. Code Optimization.
6. Graphical representation.
7. Developing standalone applications.

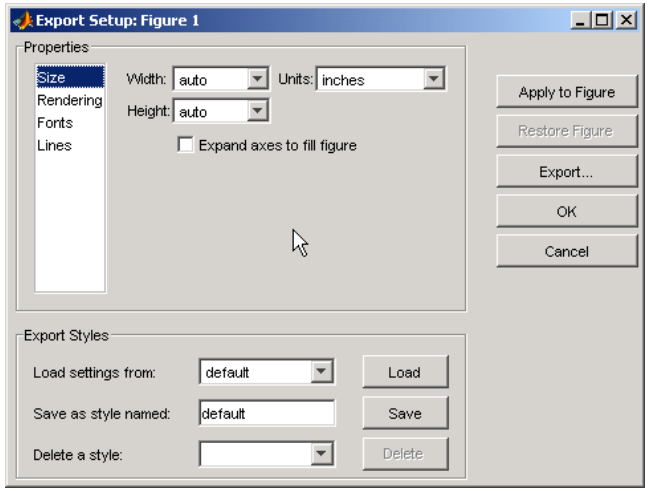
Create graphics

- Matlab can create different types of graphics:
  - to display data values
  - to create images/movies/VR/GIS
  - to generate a graphical use interface



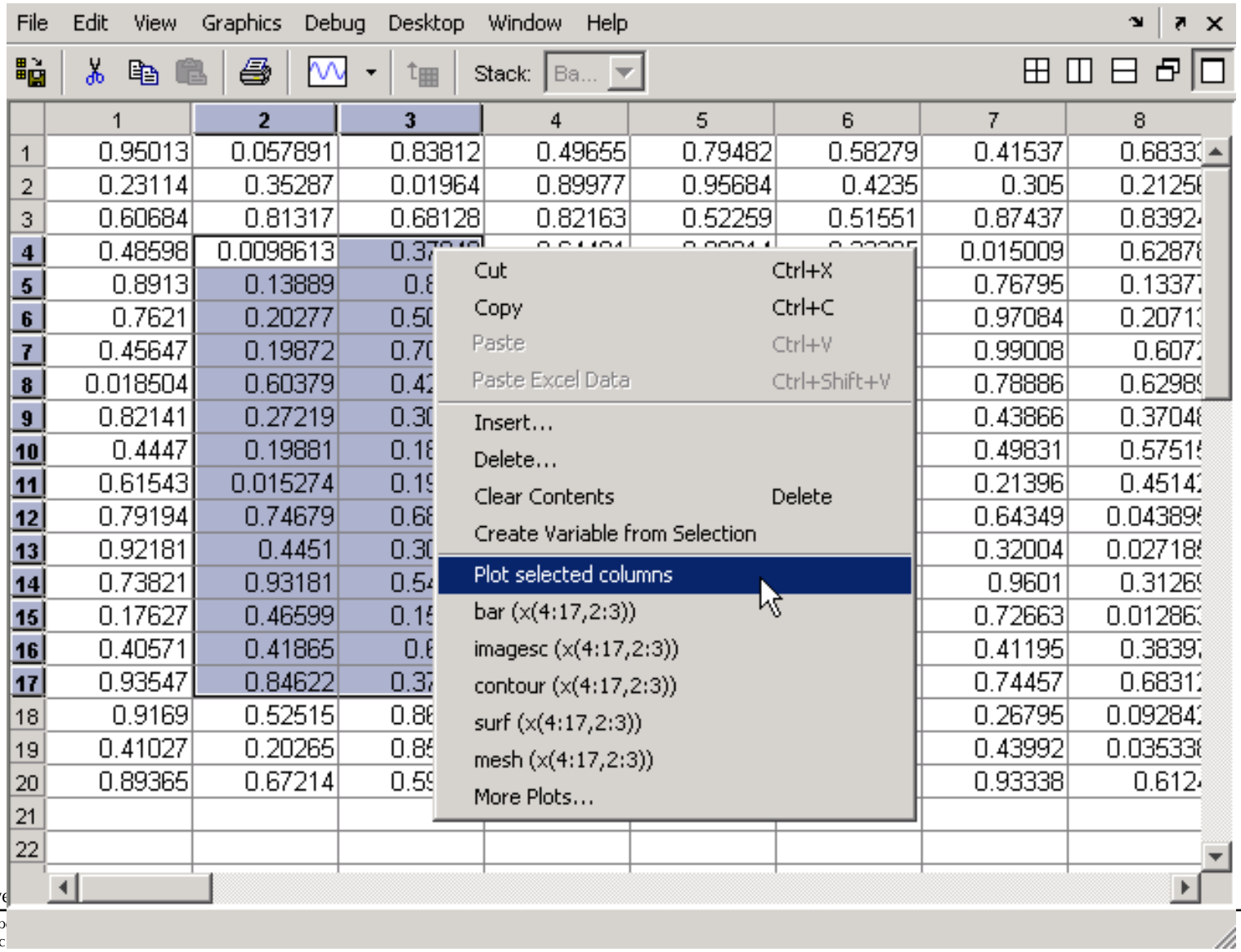




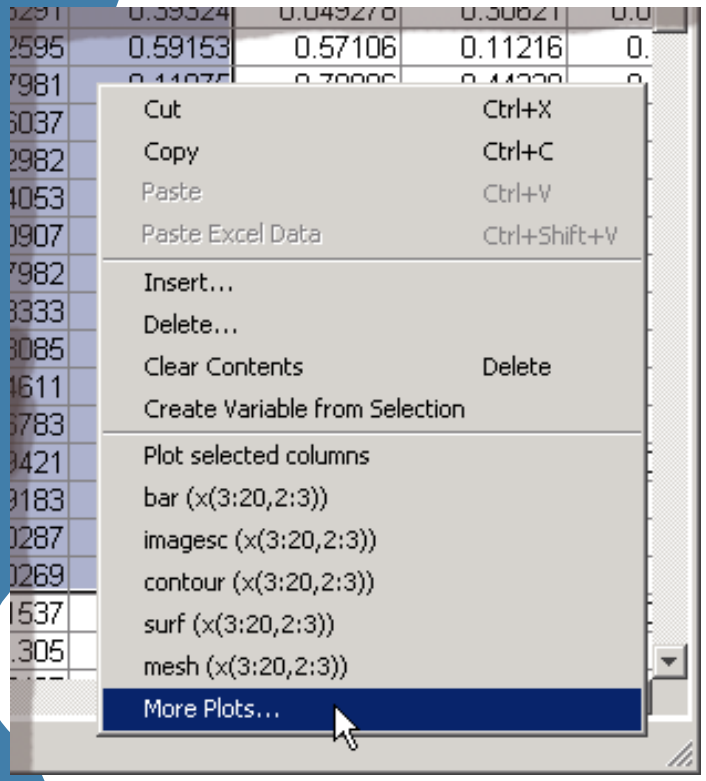


Create graphics

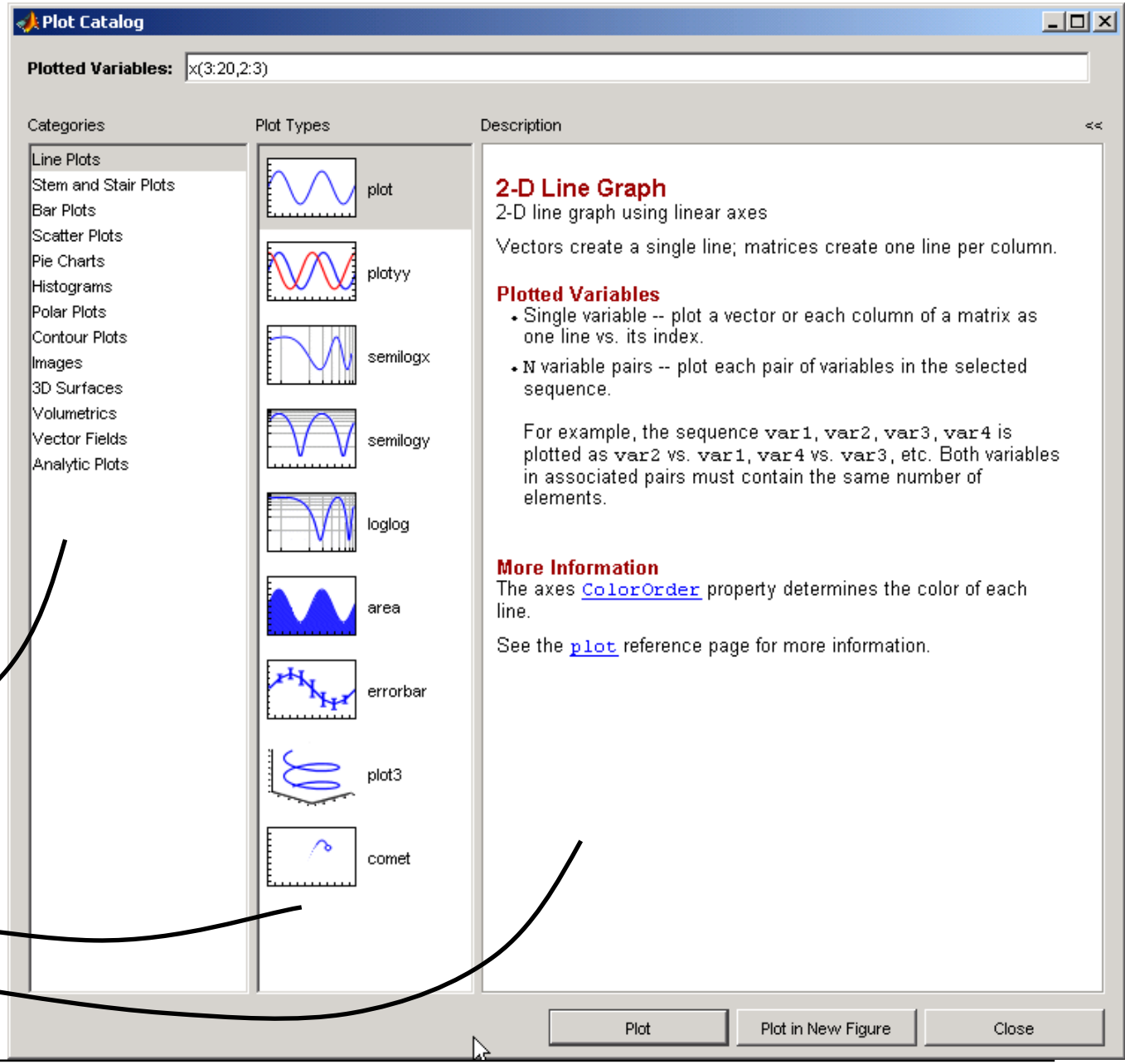
- Create graphics directly from the matrix editor



Select graphics type



Matrix editor



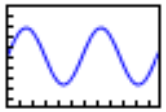
Categories

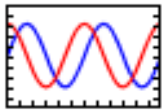
Plot types

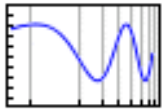
Description and function references

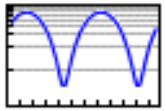
Types of graphs (1D, 2D)

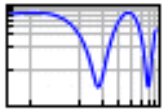
Line

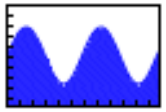

plot

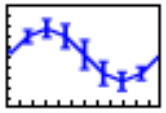

plotyy


semilogx

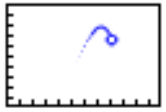

semilogy


loglog

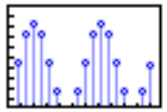

area

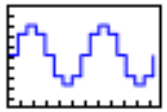

errorbar


plot3


comet

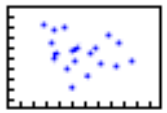
Stem & stair

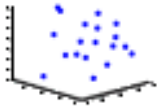

stem

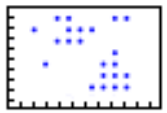

stairs

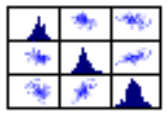

stem3

Scatter

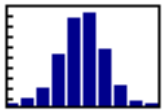

scatter


scatter3


spy

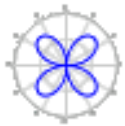

plotmatrix

Histogram


hist


rose

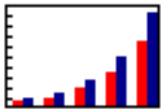
Polar

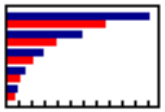

polar

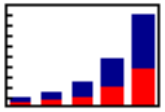

rose

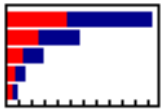

compass

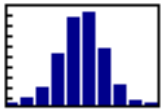
Bar

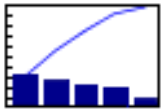

bar

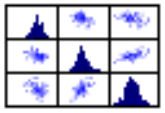

barh


bar


barh


hist


pareto


plotmatrix

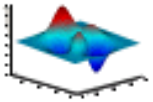
Pie

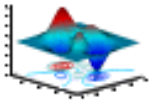

pie

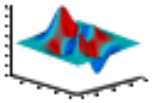

pie3

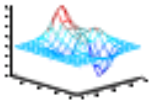
Types of graphs (>=3D)

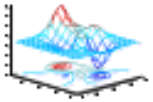
3D surfaces

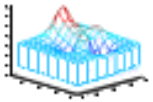

surf

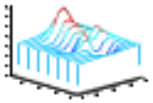

surfc


surfl


mesh


meshc

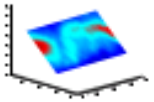

meshz

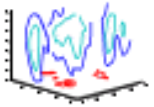

waterfall

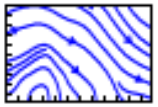

ribbon


contour3

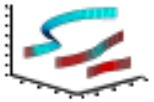
Volumetrics


slice

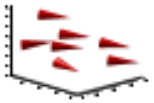

contourslice


streamslice

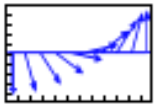

streamline


streamribbon

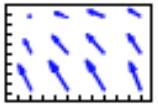

streamtube

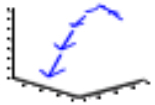

coneplot

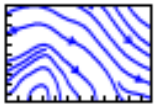
Vector Fields


feather

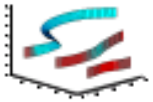

compass


quiver

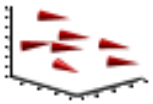

quiver3


streamslice

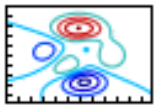

streamline

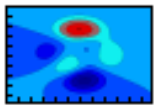

streamribbon


streamtube

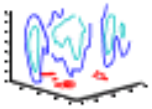

coneplot

Contour

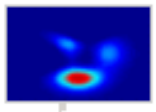

contour


contourf

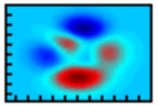

contour3

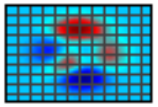

contourslice

Images

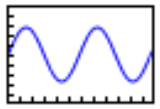


image


imagesc

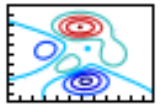

pcolor

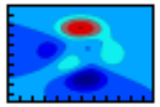
Analytic

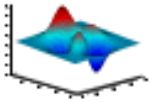

ezplot

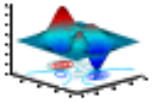

ezplot3

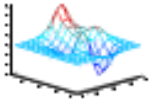

ezpolar


ezcontour


ezcontourf


ezsurf


ezsurfc

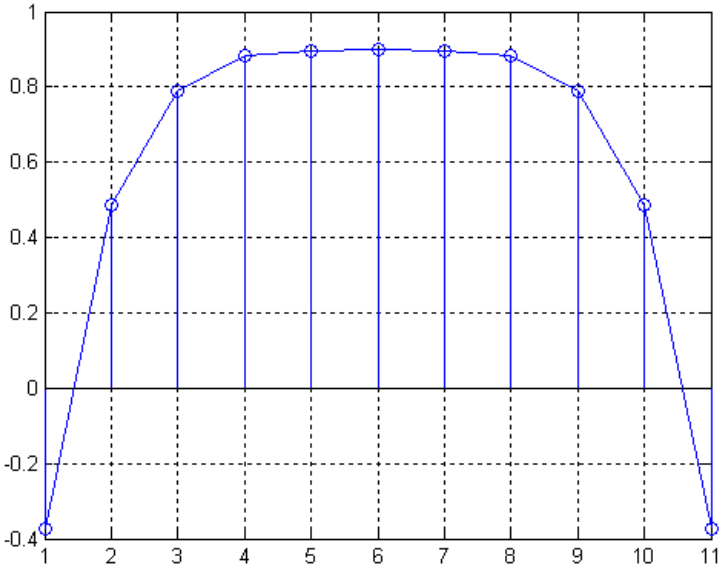

ezmesh

ezmeshc

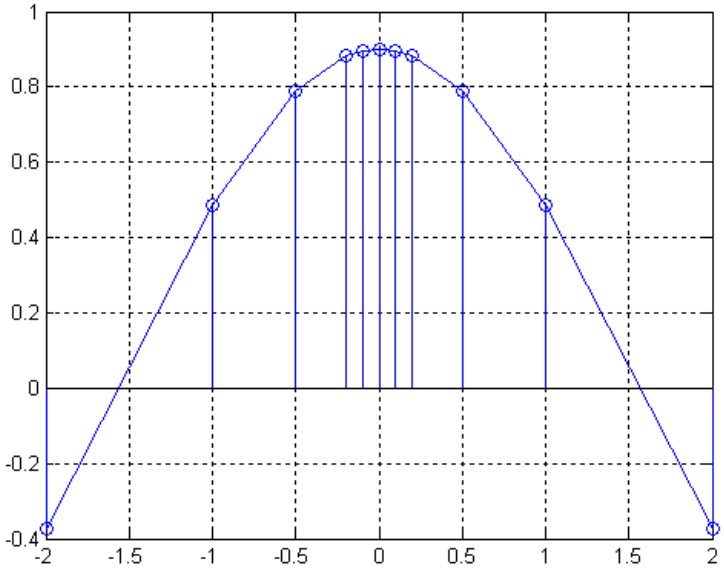
Create graphs with plot

```
plot(Y)
plot(X1,Y1,...)
plot(X1,Y1,LineSpec,...)
plot(...,'PropertyName',PropertyValue,...)
```

```
>> plot(yy,'o-')
>> hold on; stem(yy); hold off
>> grid on
```

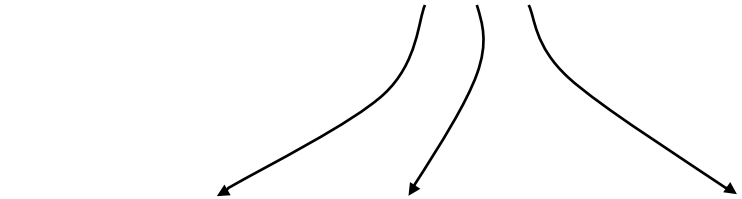



```
>> plot(xx,yy,'o-')
>> hold on; stem(xx,yy); hold off
>> grid on
```



Create graphs with plot

```
plot(x,y,'rx-');
```



y	yellow	.	punto	-	línea continua
m	magenta	o		:	línea de puntos
c	cyan	x		-.	punto y raya
r	red	+		--	línea discontinua
g	green	*			
b	blue				
w	white				
k	black				

```
plot(x,y1,'rx-',x,y2,'g--');
```

To draw a line use:

```
hold on
plot([x1,x2],[y1,y1],'k');
hold off
```

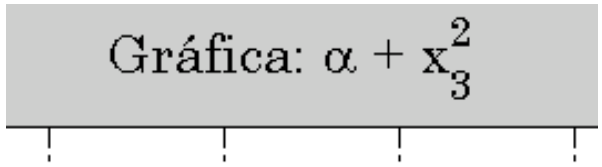
```
plot(x,y1,'rx-');
hold on;
plot(x,y2,'g--');
hold off;
```

Text on graphs

```
xlabel('Eje X');
ylabel('Eje y');
zlabel('Eje z');
title('Título de la gráfica');
text(x,y,'Texto en (x,y)');
```

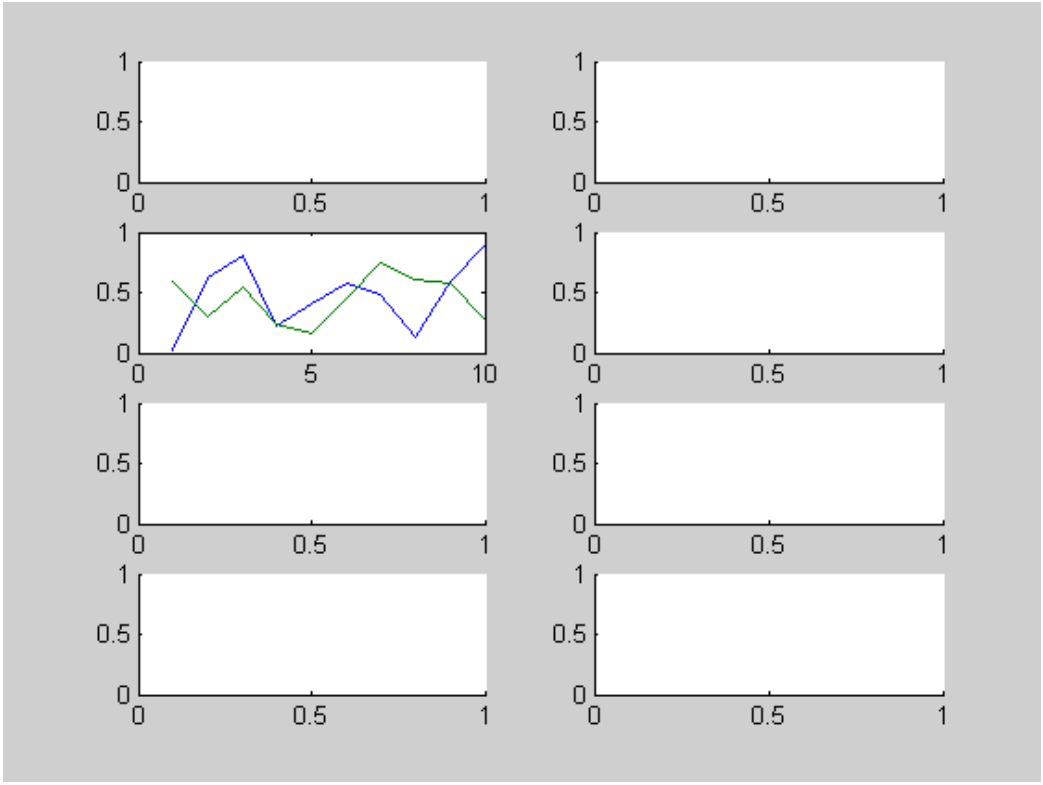
Trick: Text can use Latex expressions

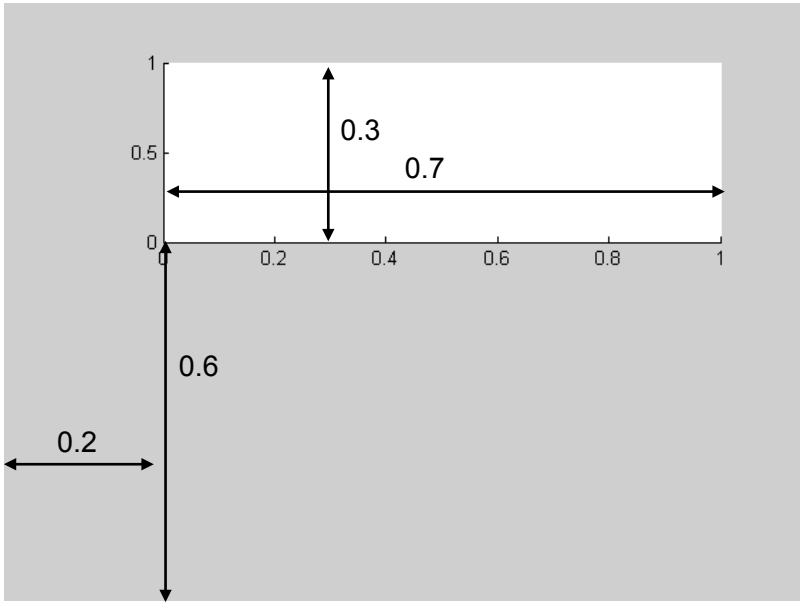
```
title('Gráfica: \alpha + x_3^2')
```



Subplots: several graphs on the same figure

```
subplot(4,2,3)
plot(rand(10,2))
```





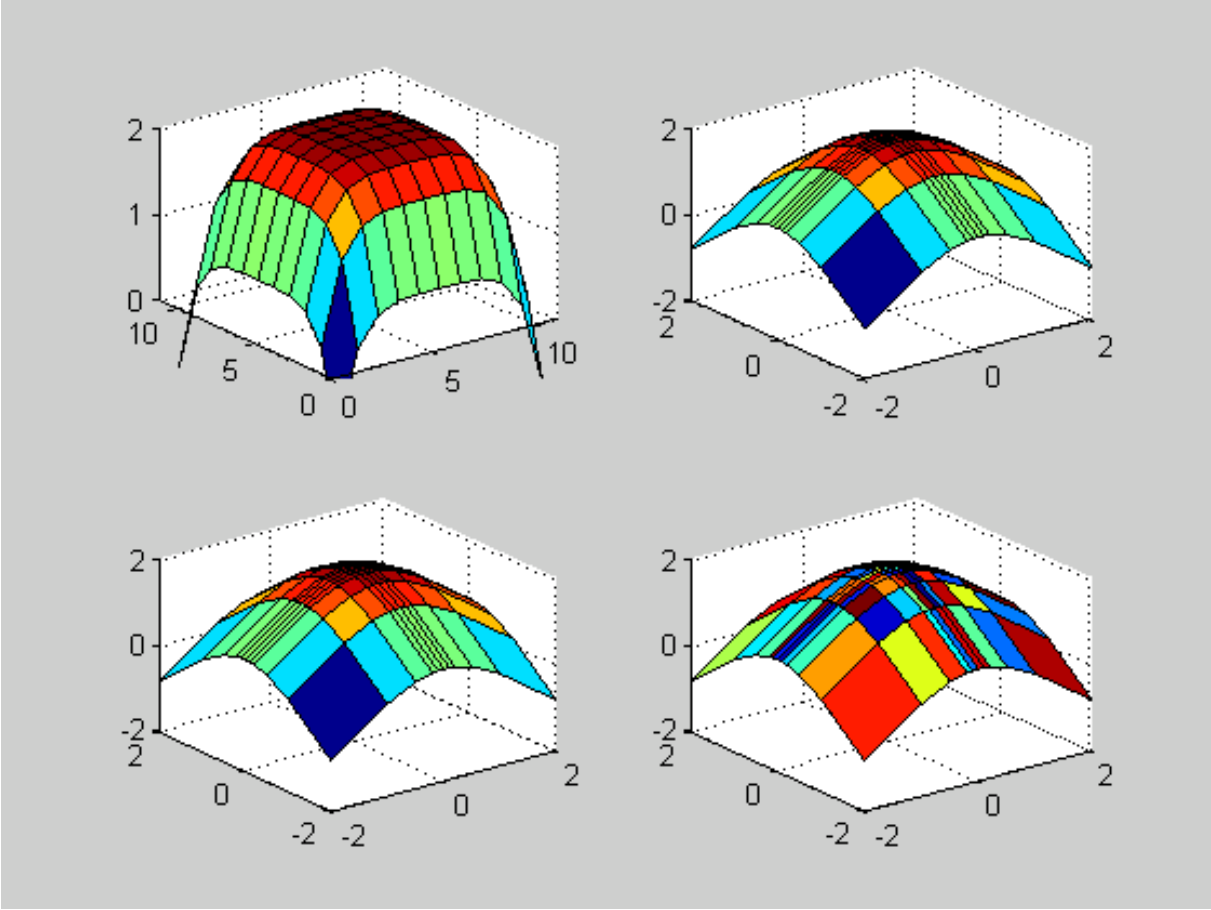

```
subplot('position',[0.2,0.6,0.7,0.3])
```

Surface graphs

```
surf(Z)
surf(X,Y,Z)
surf(X,Y,Z,C)
```

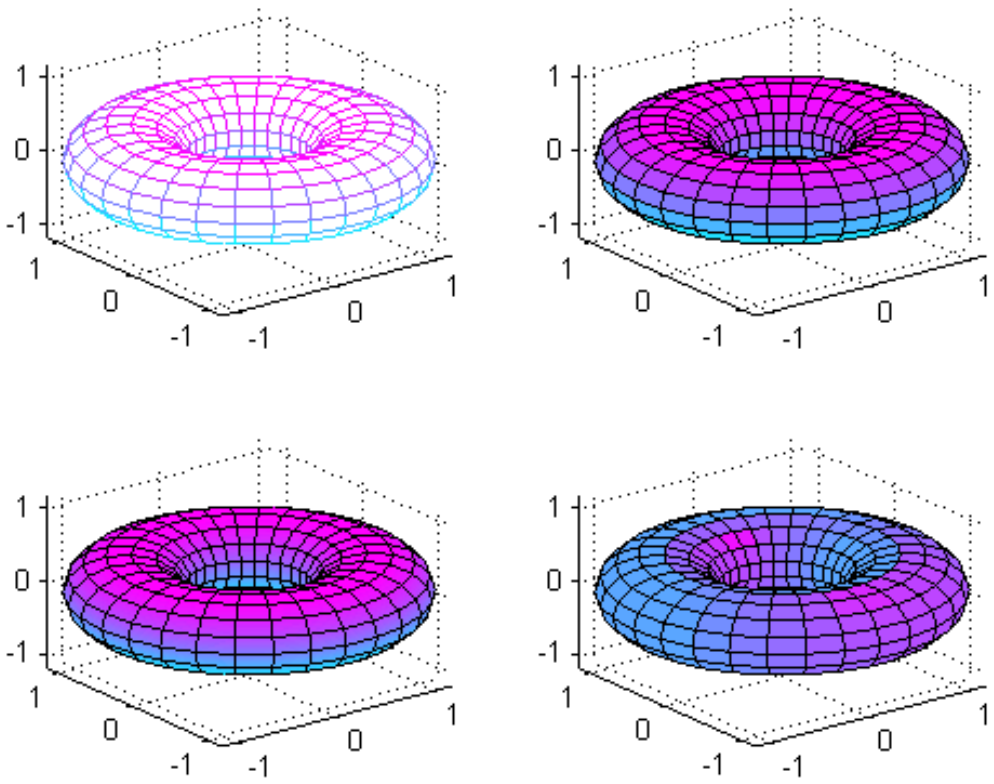
colores

```
x=[-20,-10,-5,-2,-1,0,1,2,5,10,20]/10;
y=x;

[X,Y]=meshgrid(x,y);
Z=cos(X)+cos(Y);

subplot(2,2,1)
surf(Z);
axis([0 11 0 11 0 2])

subplot(2,2,2)
surf(x,y,Z);

subplot(2,2,3)
surf(X,Y,Z);

subplot(2,2,4)
C=rand(size(Z));
surf(X,Y,Z,C);
```



Spacial surfaces

```
function [x,y,z]=torus()
% Dibuja un toro
%
r=0.5; %radio lateral
n=30; %número de elementos
a=1; %radio central

%Calculo ángulos en función de la resolución
theta=pi*(0:2:2*n)/n;
phi=2*pi*(0:2:n)'/n;

%Calculo y proyecto en x,y,z.
xx=(a + r * cos(phi))*cos(theta);
yy=(a + r * cos(phi))*sin(theta);
zz=r * sin(phi)*ones(size(theta));

%Dibujo la figura
ar=(a+r)/sqrt(2)*1.1;
colormap('cool')
subplot(2,2,1); mesh(xx,yy,zz);
axis([-ar,ar,-ar,ar,-ar,ar]);

subplot(2,2,2); surf(xx,yy,zz);
axis([-ar,ar,-ar,ar,-ar,ar]);

subplot(2,2,3); p=surf(xx,yy,zz);
shading interp
set(p,'EdgeColor','k');
axis([-ar,ar,-ar,ar,-ar,ar]);

subplot(2,2,4); surfl(xx,yy,zz);
axis([-ar,ar,-ar,ar,-ar,ar]);
```



Illumination and view point

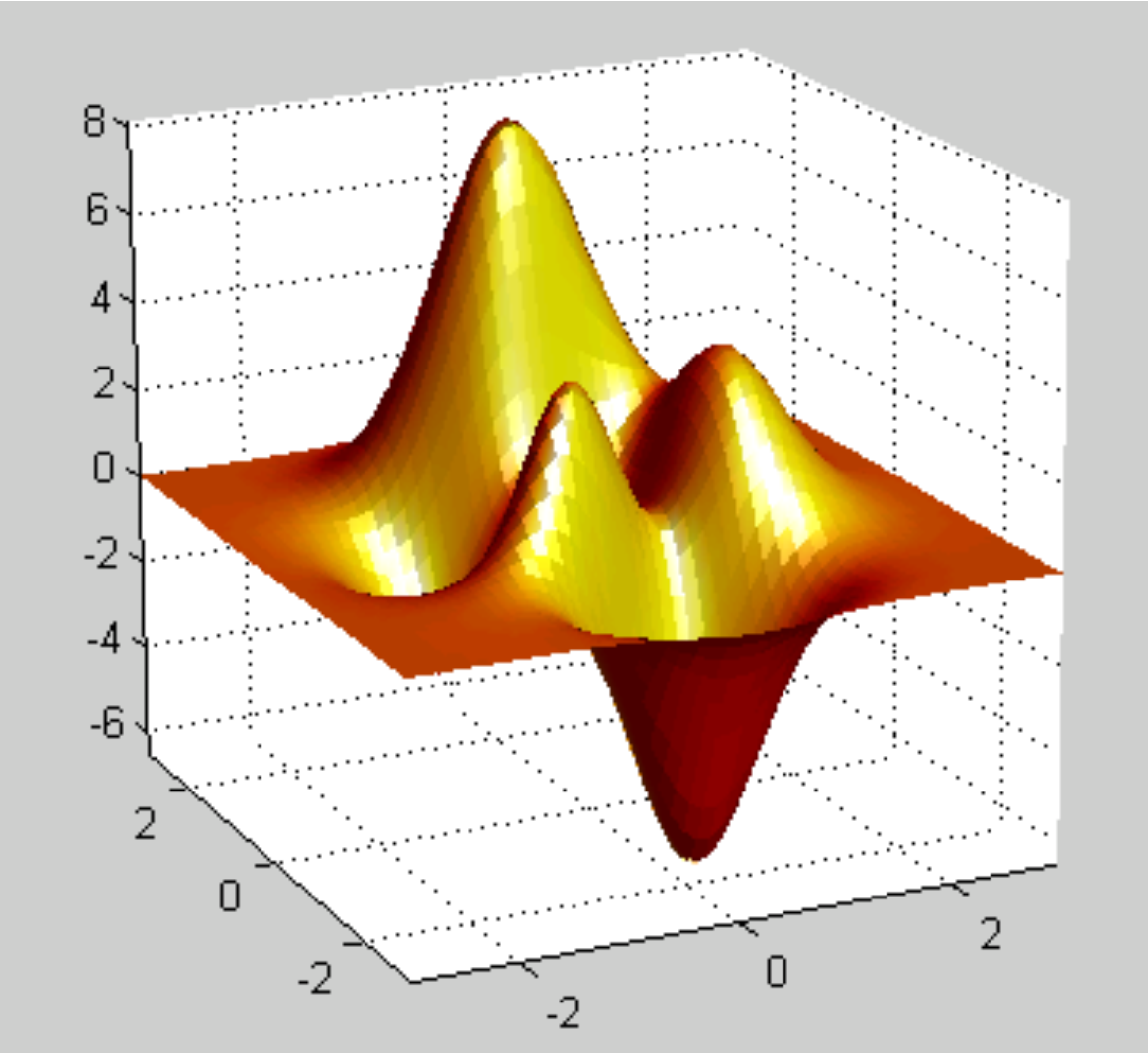
```
[x,y,z]=peaks;
surfl(x,y,z);
shading interp
```

Punto de vista

```
view(azimuth,elevation)
view(-37.5, 30)
```

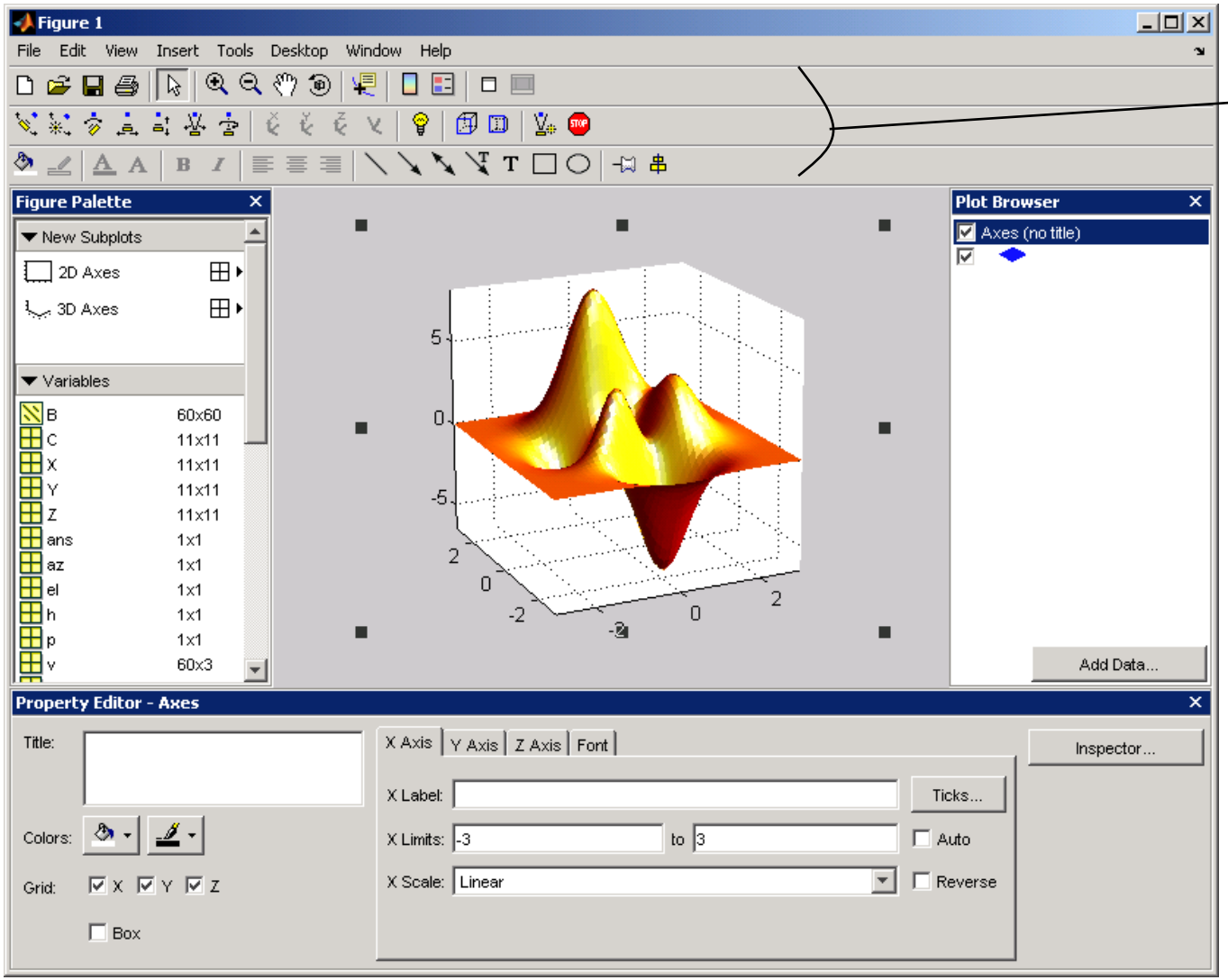
Iluminación

```
lightangle(az,el)
lightangle(90, 21.8)
```



Graph adjustments through menus

Ventana de la figura con todas las opciones activadas



Toolbars

Create new subplots

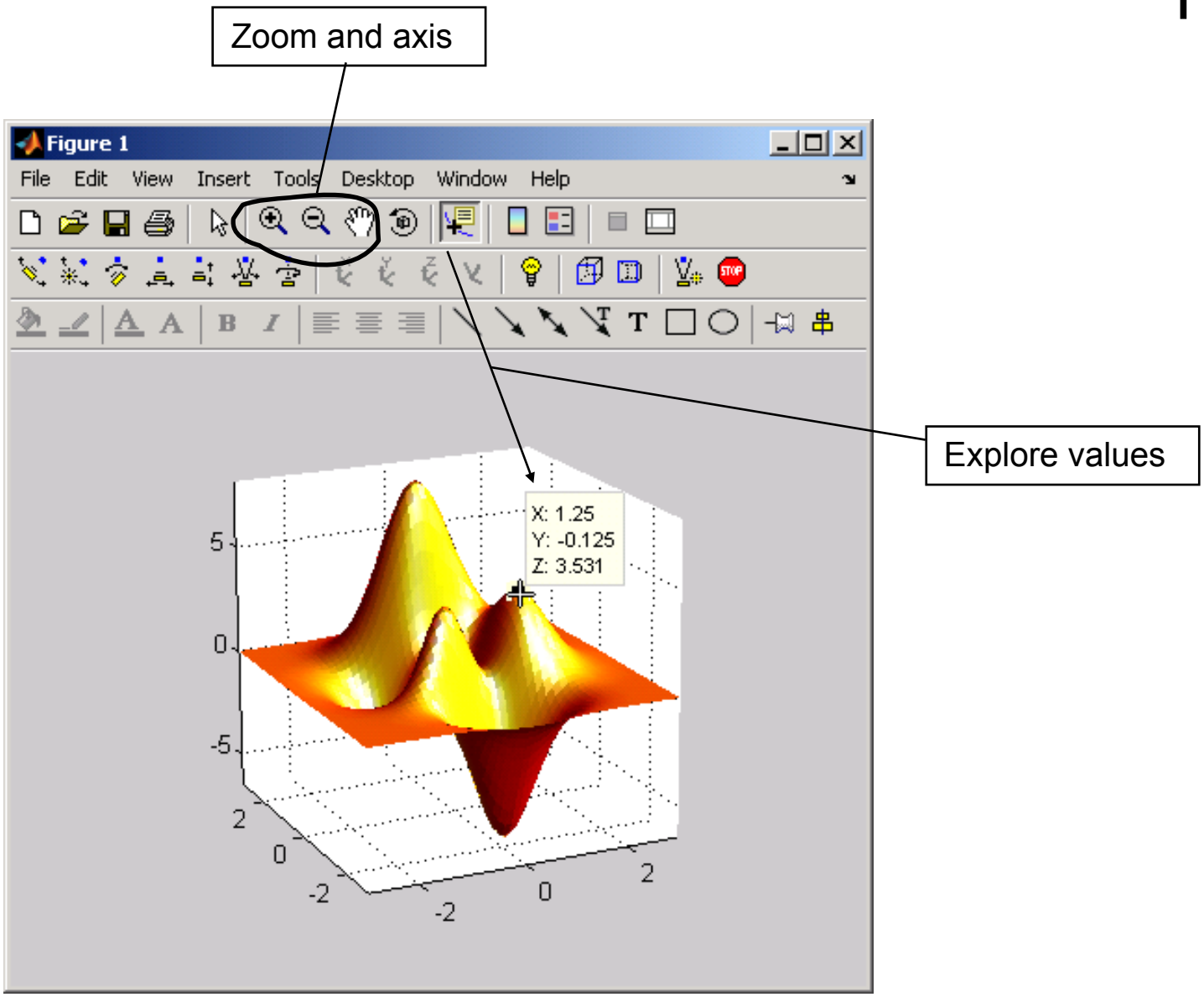
Variables of workspace

Object selector

Properties:
- Figure
- Axes
- Current Object

Graph adjustments through menus

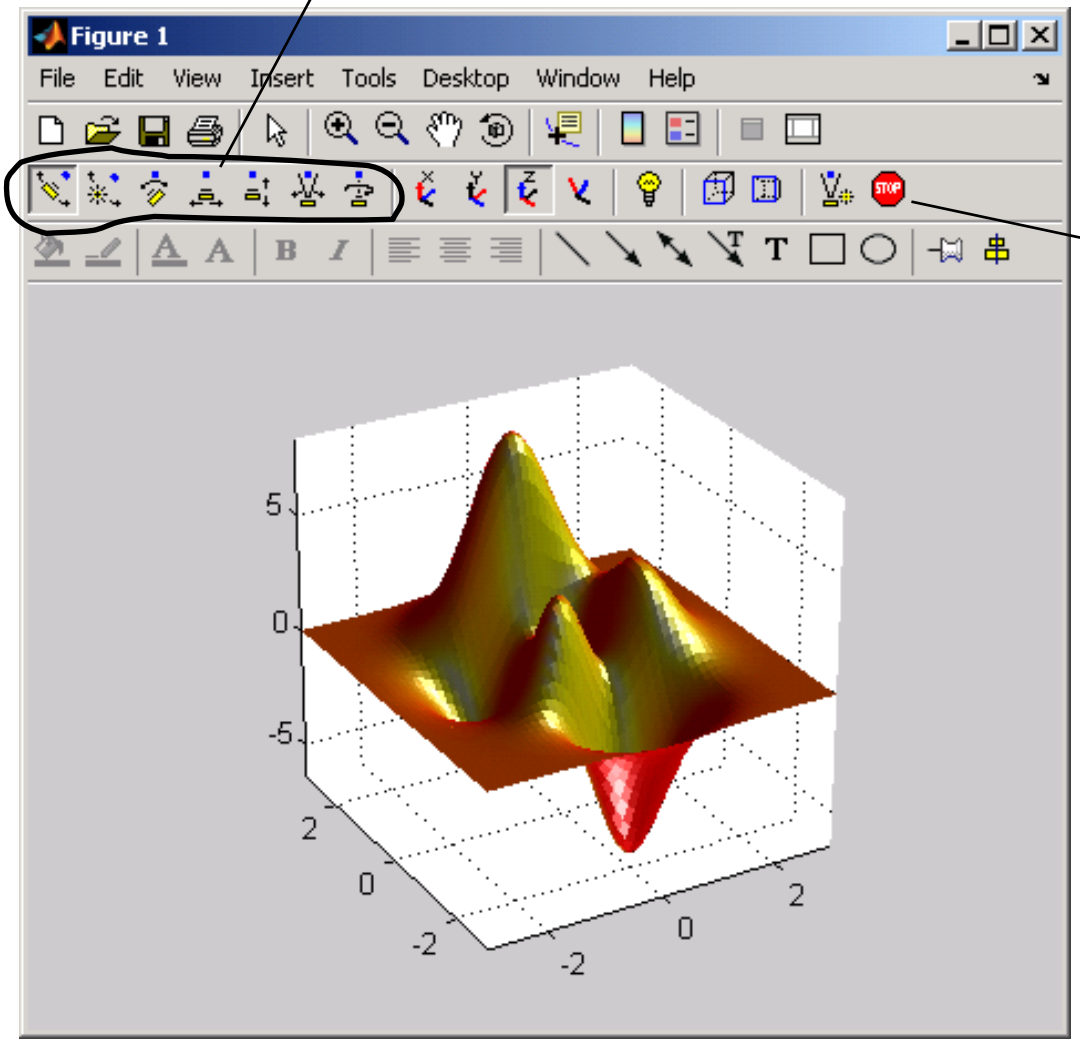
Figure Toolbar



Graph adjustments through menus

Camera Toolbar

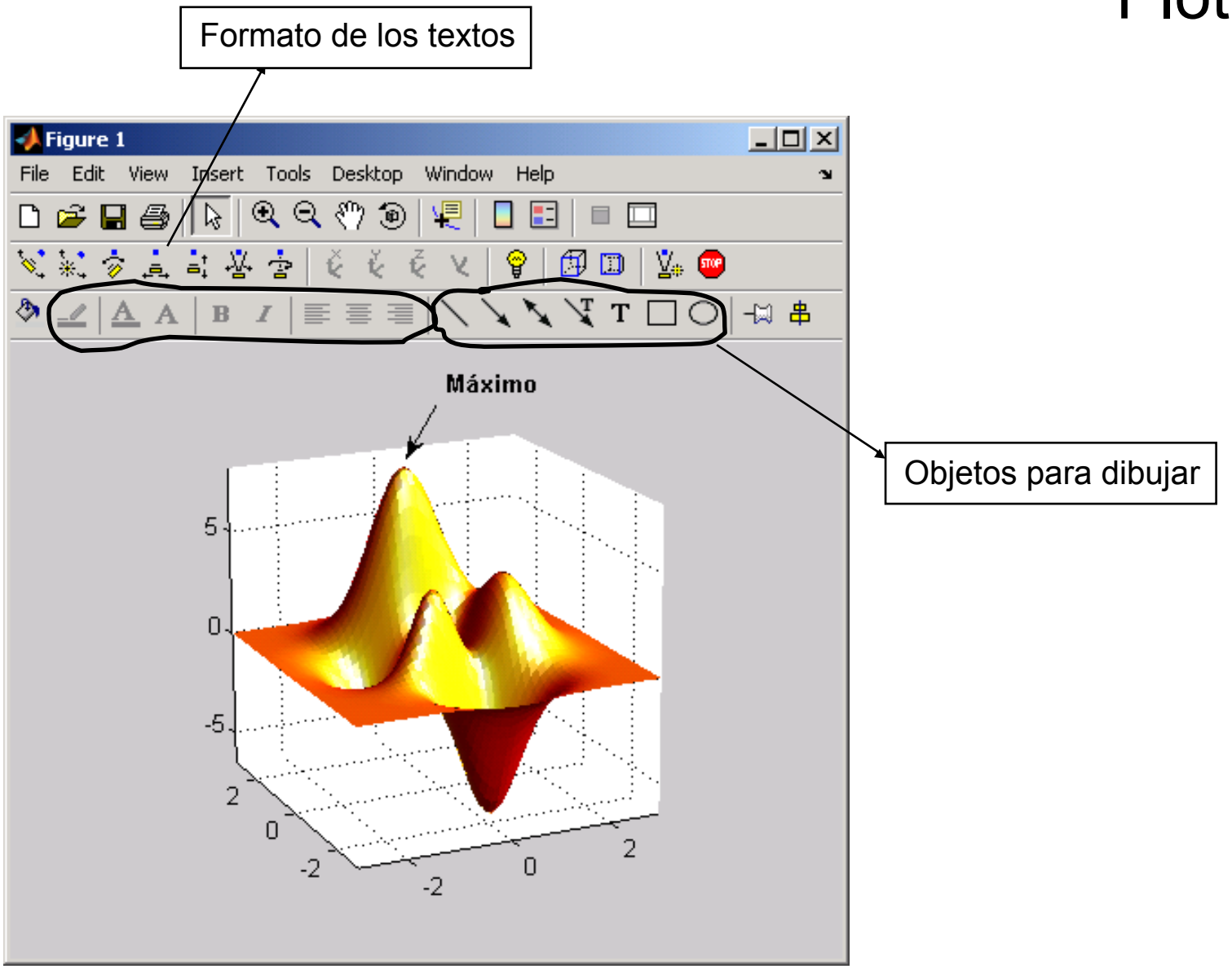
Controles de la cámara y del punto de luz



Detener la rotación animada

Graph adjustments through menus

Plot edit Toolbar



Graph adjustments through properties

- Todas las propiedades de cada objeto del gráfico (figura, ejes, elemento gráfico,...) están guardadas en "handles"
- `gcf` →current figure, `gca` →current axis
- `get(handle)` muestra todos las propiedades que se pueden cambiar
- `set(handle,'PropertyName','Value',...)` cambia propiedades

- Ejemplo:

```
set(gca,'Xtick',[1 2 3 4 5 6]);
set(gca,'XtickLabel',['ene';'feb';'mar';'abr';'may';'jun']);
```

Save figures

- Using the menu
  - File/Save As → .fig, .eps, .png, .jpeg, .bmp, .pcx, .tiff
  - File/Generate M file

    Esta opción nos permite ver qué comandos se utilizan para crear las modificaciones que hemos realizado por menú

- Using commands
  - `hgsave my_fig` → `my_fig.fig` Can be loaded using hgload
  - save the figure as an image
    - `print -depsc -tiff -r300 archivo`
    - `print –dpng –r150 archivo`

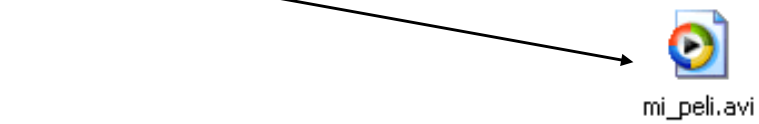
Create animations

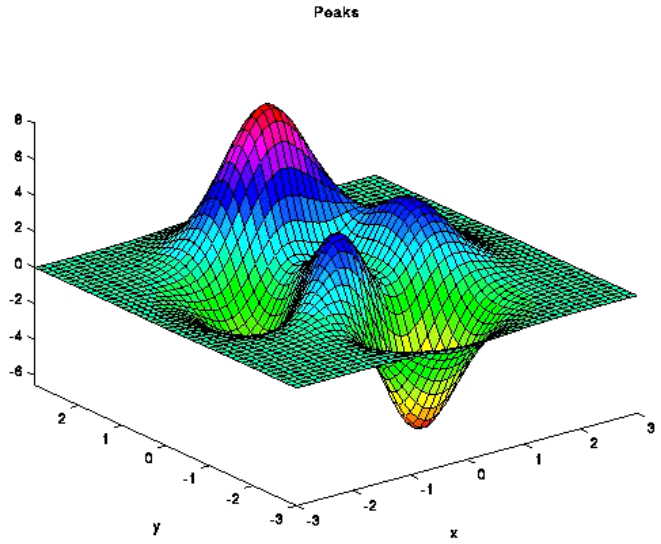
- There are two ways to create animations:
  - Offline: generate a movie, save it, watch the movie with a player.
  - On-Line: draw a sequence of graph in Matlab

- To create a movie:

  - getframe, movie

```
for k = 1:16
    plot(fft(eye(k+16)))
    axis equal
    M(k) = getframe;
end
movie(M,1); %play the movie
movie2avi(M,'mi_peli','fps',1);
```





Contents

1. Introduction to Matlab.
2. Basic data types.
3. Matlab programming.
4. Advanced data types.
5. Code Optimization.
6. Graphical representation.
7. Developing standalone applications.

Compiler

- Converts Matlab code to C and generates an executable independent from Matlab
  - The program created does not require a Matlab license to work
  - It may execute faster (not always)
  - Compiler works with functions, not scripts

Installation (only the first time)

```
mbuild -setup
```

Command to compile

```
mcc -m prueba.m
```

Compiler

- Compiler can be used to create:
  - Independent application: `mcc -m file1.m`
  - Library funcions: `mcc -l file1.m`
  - COM object (component object model)
  - Excel Add-in

Compiler

- In order to install an application in a computer without Matlab:
  - Copy the following files:

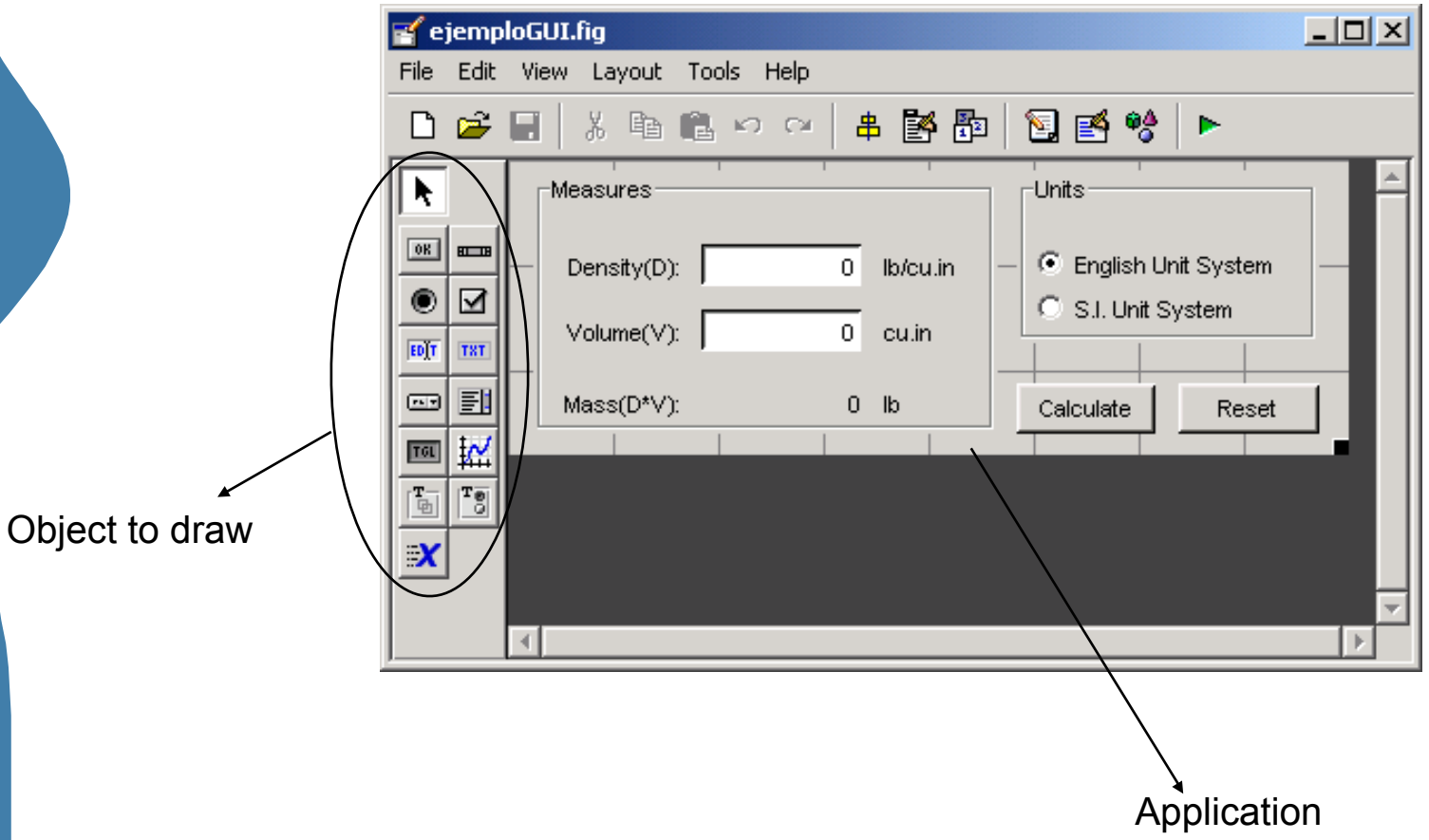
    `prueba.exe`

    `prueba.ctf`

    `<matlabroot>\toolbox\compiler\deploy\win32\MCRInstaller.exe`
  - In the destination:
    - Install `MCRInstaller` in directory `C:\MCR` (for example)
    - Check that `c:\MCR\runtime\win32` is part of PATH
    - Copy `prueba.exe` and `prueba.ctf` to any directory.

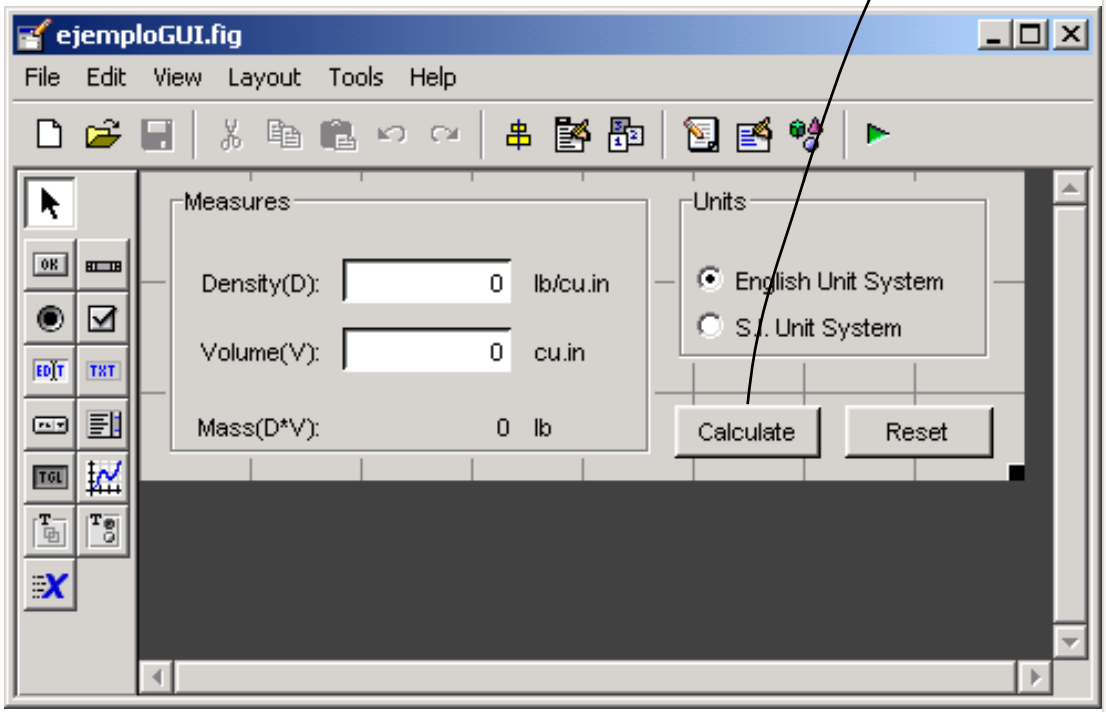
Create graphical user interfaces

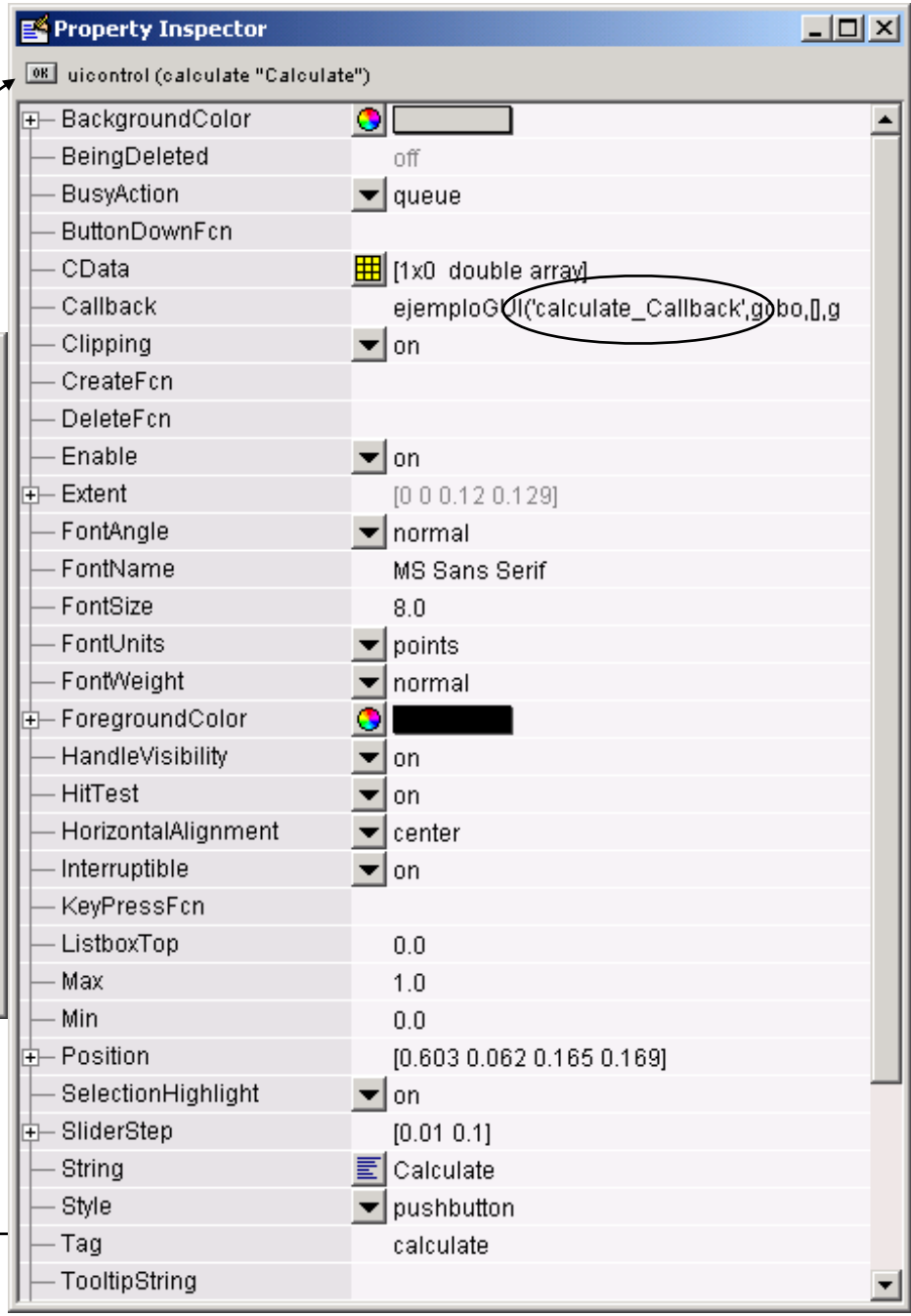
- Use `guide` from Matlab



Create graphical user interfaces

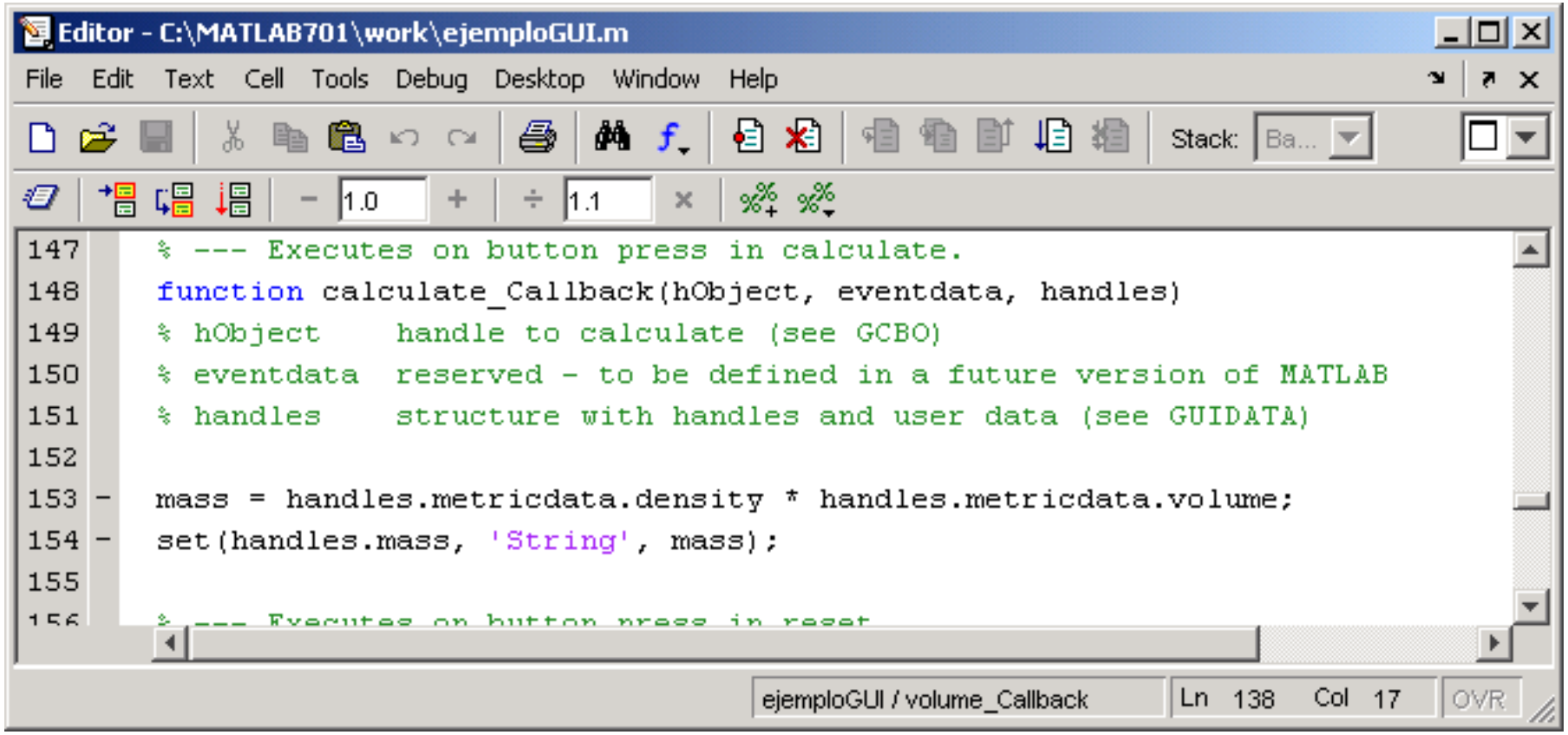
- Each object has some attributes and a callback function





Create graphical user interfaces

- Guide generates a .m file ready to insert your application code



In general it is recommended to keep computing code isolated from interface functions

External communications

- Direct data acquisition
  - Database toolbox
  - Data Acquisition toolbox
  - Image Acquisition toolbox

- Data from files
  - xlsread, load, textscan (numbers)
  - auread, wavread (sound)
  - imread (image)
  - aviread (movie)

External communications

- External programs
  - Matlab calls other programs
    - `system`, `dos`, `unix`
  - Matlab calls functions developed in different programming languages
    - Create a MEX (#include "mex.h")
    - Matlab call the function is if it were a .m
  - Excel calls matlab
    - Excel Link toolbox
  - Any program calls matlab avoiding the interface
    - `matlab -nodisplay -m programa`
    - `matlab -nodesktop -m programa`

Generate documentation

- Useful recommendations
  - Document functions:
    - Description
    - Input arguments
    - Output results
    - Example on how to use the function
    - Remarks/limitations
  - Keep you computing code independent of the interface code. Better for debugging, improving performance and to update the interface.
  - Create sections with comment of type %%

Generate documentation

- Matlab 7 and latter can generate documentation automatically.
  - Generates HTML, XML, LaTeX, Word y Power Point.
  - It uses the comments %%

- Procedure:
  - Activate "cell mode" with Cell/Enable Cell Mode
  - Select File/Publish to HTML
  - Matlab will execute the script and will generate a document with your code and the graphical results created on each section.

Ejemplo: script torus.m

Editor - C:\MATLAB701\work\torus.m

```
%convierto a script para generar documentación
%function [x,y,z]=torus()
% Dibuja un toro
%
r=0.5; %radio lateral
n=30; %número de elementos
a=1; %radio central

%% Calculo ángulos en función de la resolución
theta=pi*(0:2:2*n)/n;
phi=2*pi*(0:2:n)'/n;

%% Calculo y proyecto en x,y,z.
xx=(a + r * cos(phi))*cos(theta);
yy=(a + r * cos(phi))*sin(theta);
zz=r * sin(phi)*ones(size(theta));

%% Dibujo mesh
ar=(a+r)/sqrt(2)*1.1;
colormap('cool')
subplot(2,2,1); mesh(xx,yy,zz); axis([-ar,ar,-ar,ar
%% Dibujo Superficie
subplot(2,2,2); surf(xx,yy,zz); axis([-ar,ar,-ar,ar
%% Dibujo con interpolación de colores
subplot(2,2,3); p=surf(xx,yy,zz);
shading interp
set(p,'EdgeColor','k');
axis([-ar,ar,-ar,ar,-ar,ar]);
%% Dibujo con iluminación
subplot(2,2,4); surfl(xx,yy,zz); axis([-ar,ar,-ar,a
```

ejemploGUI.m | calculo.m | torus.m

torus.doc - Microsoft Word

Calculo ángulos en función de la resolución 1
Calculo y proyecto en x,y,z 1
Dibujo mesh 1
Dibujo Superficie 2
Dibujo con interpolación de colores 3
Dibujo con iluminación 4

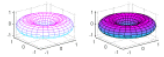
Calculo ángulos en función de la resolución

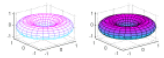
Calculo y proyecto en x,y,z.

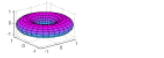
Dibujo mesh

Dibujo Superficie

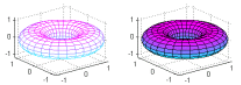
Dibujo con interpolación de colores

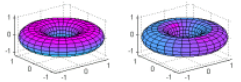






Dibujo con iluminación





On-Line resources

- **Mathworks website**
  `http://www.mathworks.com/support/`
  - Documentation
  - Support. Sorted by categories
  - Sample code
  - News
  - Software updates
- **Matlab Central**
  - Newsgroups
  - File Exchange
  - Link Exchange
- **Technical support by email**
  - You need to provide your active license code.
  - You must describe platform, operating system, etc.
  - Be very specific with your problem.